Intel Corporation
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-6497

(503) 696-8080

June 1994

Dear Paragon™ Customer:

The Paragon™ XP/S supercomputer can accommodate thousands of advanced microprocessors connected in a two-dimensional rectangular mesh. The Paragon XP/E supercomputer is a distributed-memory multicomputer that can accommodate up to thirty advanced compute nodes connected in a two-dimensional rectangular mesh.

The interconnect network offers high-bandwidth, low-latency communication and frees programmers from having to concern themselves with interconnect topology. The Paragon operating system brings the Open Software Foundation's industry standard version of the UNIX operating system to the performance-driven environment of scalable, distributed-memory computing.

Thank you for joining the fast-growing community of scientists and programmers now taking advantage of Paragon systems to tackle problems not before possible at an affordable price.

This package contains the system software for the Paragon system. Please read through the documentation and distribute the materials to those intending to use the system.

Before using your system:

- **Read this letter completely.**
- **Verify the contents of this package.**
- **Read the *Paragon™ System Software Release 1.2 Release Notes.***

Package Contents

The operating system software is provided on two 0.25-inch 12000 BPI cartridge tapes in UNIX tar format.

Table 1. Installation Media

Description	Order Number
Cartridge tape labeled Paragon™ System Software Release 1.2	313095-001
Cartridge tape labeled Paragon™ SAT Software Release 1.2	313097-001

Diagnostic and compiler software are packaged separately If you are a brand new customer, receiving Paragon operating system software for the very first time, Table 2 lists the included documentation.

Table 2. Documentation (1 of 3)

Description	Order Number
Paragon™ System Software Release 1.2 Release Notes	313057-001
User Group Membership Pack	312805-001
OSF/1 Documentation Errata	312857-002
Paragon™ System Technical Documentation Guide	312820-002
Paragon™ Commands Reference Manual	312486-003
Paragon™ User's Guide	312489-003
Paragon™ Interactive Parallel Debugger Reference Manual	312547-003

Table 2. Documentation (2 of 3)

Description	Order Number
Paragon™ Application Tools User's Guide	312545-003
Paragon™ System Acceptance Test User's Guide	312648-003
Paragon™ Network Queueing System Manual	312645-002
Paragon™ Graphics Libraries User's Guide	312887-001
Paragon™ System Performance Visualization Tool User's Guide	312889-002
Paragon™ High-Performance Parallel Interface Manual	312824-002
Paragon™ XP/S i860™ 64-Bit Microprocessor Assembler Reference Manual	312546-001
i860™ Microprocessor Family Programmer's Reference Manual	240875-002
Paragon™ Site Preparation Guide	312485-003
Paragon™ Hardware Installation Manual	312543-003
Paragon™ System Administrator's Guide	312544-003
Paragon™ XP/S RAID Utilities Manual	312646-003
Paragon™ Hardware Maintenance Manual	312822-001
Paragon™ System Cabling Guide	312823-001
Paragon™ Multi-User Accounting and Control System Utilities Manual	312891-002
X Protocol Reference Manual	312654-001
Xlib Programming Manual	312655-001
Xlib Reference Manual	312656-001
X Toolkit Intrinsics Reference Manual	312658-001
X Toolkit Intrinsics Programming Manual	312657-001
Open Desktop User's Guide	312954-001
Open Desktop Administrator's Guide	312955-001

Table 2. Documentation (3 of 3)

Description	Order Number
OSF/1 Network Application Programmer's Guide	PSCOSF1M*
OSF/1 Command Reference	PSCOSF1M†
OSF/1 Programmer's Reference	PSCOSF1M†
OSF/1 System and Network Administrator's Reference	PSCOSF1M†
OSF/1 User's Guide	PSCOSF1M†

*This product code designates all five volumes of the OSF/1 documentation set.

If you are an existing customer, receiving an update, Table 3 lists the included documentation. This list includes all manuals and release notes that have changed since your last shipment.

Table 3. Updated Documentation (1 of 2)

Description	Order Number
Paragon™ System Software Release 1.2 Release Notes	312927-001
Paragon™ Site Preparation Guide	312485-003
Paragon™ Commands Reference Manual	312486-003
Paragon™ C System Calls Reference Manual	312487-003
Paragon™ Fortran System Calls Reference Manual	312488-003
Paragon™ User's Guide	312489-003
Paragon™ Hardware Installation Manual	312543-003
Paragon™ System Administrator's Guide	312544-003
Paragon™ Application Tools User's Guide	312545-003
Paragon™ Interactive Parallel Debugger Reference Manual	312547-003
Paragon™ XP/S RAID Utilities Manual	312646-003
Paragon™ Network Queueing System Manual	312645-003
Paragon™ System Acceptance Test User's Guide	312648-003

Table 3. Updated Documentation (2 of 2)

Description	Order Number
Paragon™ System Technical Documentation Guide	312820-002
Paragon™ Hardware Maintenance Manual	312822-001
Paragon™ System Cabling Guide	312823-001
Paragon™ High-Performance Parallel Interface Manual	312824-002
OSF/1 Documentation Errata	312857-002
Paragon™ System Performance Visualization Tool User's Guide	312889-002
Paragon™ Multi-User Accounting and Control System Utilities Manual	312891-002

Additional materials are listed in the letters describing the diagnostics, compiler, and tools software that accompany Release 1.2.

If you purchased a Paragon XP/E supercomputer, you also receive the manuals listed in Table 4. Use these manuals instead of the *Paragon™ Site Preparation Guide* and the *Paragon™ Hardware Installation Manual.*

Table 4. Additional Documentation for Paragon™ XP/E Systems

Description	Order Number
Paragon™ XP/E Site Preparation Guide	312842-002
Paragon™ XP/E Hardware Maintenance Manual	312843-002

If items are missing, or if you have any questions, please contact Intel Supercomputer Systems Division immediately. Please refer to the section "Comments and Assistance" in this letter for information about how to contact Intel Supercomputer Systems Division.

What is in Release 1.2?

The Paragon Release 1.2 software is the latest release of the Paragon operating system. It includes the X Window System, online manual pages, and manuals in both hardcopy and PostScript format. For a list of features in this release, refer to Chapter 1 in the *Paragon™ System Software Release 1.2 Release Notes*.

Installation

For directions on how to install the Paragon Release 1.2 software, refer to Chapter 2 in the *Paragon™ System Software Release 1.2 Release Notes.*

NOTE

Adding or removing any boards or components from your Paragon system can damage the system and may invalidate your warranty. Please contact Intel Supercomputer Systems Division Customer Support for assistance in answering your questions.

Restrictions and Limitations of Release 1.2

Every effort has been taken to ensure the quality of this release, but at shipment we are aware of some limitations. Please refer to the *Paragon™ System Software Release 1.2 Release Notes* for known limitations and available workarounds.

Comments and Assistance

We are eager to hear of your experiences with the Paragon system. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

Intel Corporation Italia s.p.a.
Milanofiori Palazzo
20090 Assago
Milano
Italy
1678 77203 (toll free)

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Intel Japan K.K.
Supercomputer Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

United Kingdom Intel Corporation (UK) Ltd.
Supercomputer System Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056 (*answered in French*)
(44) 793 431062 (*answered in Italian*)
(44) 793 480874 (*answered in German*)
(44) 793 495108 (*answered in English*)

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Supercomputer Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 629-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 629-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com (Internet)

Intel Supercomputer Users' Group

The Intel Supercomputer Users Group promotes the exchange of information among users. Intel strongly supports the Users Group and encourages participation in its activities, which include: Special Interest Groups (SIGs), an annual international users conference, an electronic mail task force, and a "freeware" library of user-contributed software, available electronically to all members of the Intel Supercomputer Users' Group. For membership information contact:

JoAnne Wold (503-629-5322)
joanne@ssd.intel.com (Internet)

Again, thank you for acquiring a Paragon Supercomputer.

Sincerely,

Steve Cannon

Product Marketing Manager
Intel Supercomputer Systems Division

Paragon is a trademark of Intel Corporation.
UNIX is a trademark of UNIX System Laboratories.
SCO and OPEN DESKTOP are trademarks of The Santa Cruz Operation, Inc.
The X Window System is a trademark of the Massachusetts Institute of Technology.

June 1994

Order Number: 313057-001

Paragon™ System Software

Release 1.2

Release Notes

Intel® Corporation

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

LIMITED RIGHTS

Preface

These release notes provide the latest information about features, installation, open bugs, and fixed bugs in Release 1.2 of the Paragon™ system software.

These release notes assume you are familiar with the Paragon operating system and system administration.

For more information, refer to the Paragon system documentation set and the online manual pages.

Organization

Chapter 1	Describes the new features for this release of the Paragon system software.
Chapter 2	Provides installation information for the Paragon system software.
Chapter 3	Provides PFS Performance information.
Chapter 4	Provides memory usage information and suggested guidelines for the use of this release of the Paragon system software. It also includes known limitations and workarounds, a list of the open bugs and a list of the bugs fixed in this release of the Paragon system software.

Notational Conventions

This manual uses the following notational conventions:

Bold Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.

Italic Identifies variables, filenames, directories, processes, user names, and writer annotations in examples. Italic type style is also occasionally used to emphasize a word or phrase.

`Plain-Monospace`
Identifies computer output (prompts and messages), examples, and values of variables. Some examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style and flush with the right margin.

`Bold-Italic-Monospace`
Identifies user input (what you enter in response to some prompt).

`Bold-Monospace`
Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example:

`<Break>` **`<s>`** **`<Ctrl-Alt-Del>`**

[ ] (Brackets) Surround optional items.

. . . (Ellipses) Indicate that the preceding item may be repeated.

| (Bar) Separates two or more items of which you may select only one.

{ } (Braces) Surround two or more items of which you must select one.

Applicable Documentation

For information about the manuals shipped with the Paragon system, see the *Paragon™ System Technical Documentation Guide.*

Comments and Assistance

If you are part of the Release 1.2 program, contact the Parallel Systems Engineer or Field Applications Engineer associated with your site.

Intel Supercomputer Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

Intel Corporation Italia s.p.a.
Milanofiori Palazzo
20090 Assago
Milano
Italy
1678 77203 (toll free)

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

Intel Japan K.K.
Supercomputer Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

United Kingdom Intel Corporation (UK) Ltd.
Supercomputer System Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056 (*answered in French*)
(44) 793 431062 (*answered in Italian*)
(44) 793 480874 (*answered in German*)
(44) 793 495108 (*answered in English*)

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Supercomputer Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 629-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 629-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com

Table of Contents

Chapter 1
Product Features

Chapter 2
Software Installation

Chapter 3
Parallel File System Performance

Chapter 4
Guidelines and Limitations

List of Illustrations

List of Tables

Product Features 1

Introduction

This chapter lists the major features of Paragon™ Release 1.2 system software.

NOTE

After installing this release of the Paragon system software, you must recompile and relink all your C and Fortran applications. Use the system libraries supplied with this release and the latest versions of the compilers to do this.

Features in this Release

This release of the Paragon system software includes the following features:

- The message co-processor is enabled. Message passing bandwidth is now 90M bytes/sec, and message passing latency is 46 microseconds. Use of the message coprocessor requires a hardware upgrade.
- A new allocator configuration file gives the system administrator control over the level of gang-scheduling support in the system.
- Limited support for Pthreads. The *libpthreads.a* and *libc_r.a* libraries are supported. See the *Paragon™ User's Guide* for more information.
- Automated paging tree setup. The **reset** script will automatically configure a two-level paging tree where compute nodes page to I/O nodes which page to the boot node. This paging tree is not configured if additional I/O nodes are unavailable.
- The **tar** command now supports Parallel File System (PFS) files greater than 2G bytes.

- Ability to enable space sharing on a partition basis.
- Ability to specify rectangular node sets when loading applications.
- Enhanced **showpart** now indicates currently unused nodes, and **pspart** has also been enhanced to be recursive and to display more information. lspart displays an extra column that shows the number of unused (free) nodes in each partition.
- New call **nx_app_rect**() that returns the rectangular dimensions of a partition
- A number of library calls now provide access to partition attributes. These calls facilitate tools development.
- Ability to boot with alternate operating systems on compute nodes.
- Adaptive message buffering. If the application actually communicates with more correspondents than are assigned message buffers, the operating system reassigns message buffers to allow the application to continue without using additional memory.
- The message handler (the handler established with the **h...**() calls) now run concurrently with the main program instead of interrupting the main program until the handler completes.

CAUTION

You must use **masktrap()** around any code in the main program that could interfere with calls in the handler.

Because it is often not obvious which calls could interfere with each other, use **masktrap()** to protect *all* library calls in the rest of the program that could call the same subsystems as the calls in the handler while the handler is active.

- **setptype**() is now allowed only if the current ptype is INVALID_PTYPE (processes cannot change their *ptype*).
- MACS account attributes now include number of nodes, CPU time limit, weight flag, and lock flag.
- MACS user attributes now include number of nodes, CPU time limit, and percent allocation.
- MACS has the following new commands: **macadmin**, **macupdate**, **macalloc**, and **maclist**

- NQS enhancements include a move request, soft and hard user limits, time specification in minute increments, and an option to force NQS jobs to not overrun into prime time, and support for nodes with different amounts of memory.

- NQS has the following new QMGR subcommands: **move request**, **set softulimit**, **set hardulimit**, and **set node_group**.

- New NQS *sched_param* parameters include the following: *np_overrun*, *node_set*, *node_group*, *prime_list*, and *nprime_list*. Obsolete *sched_param* parameters include the following: *majorp_nodef*, *minorp_nodef*, *timesh_minor*, and *block_ts_pri*.

- Reference manual pages have been updated and revised. Manual pages new for Release 1.2 include the following:

  - Manual pages for the boot configuration files: *BADNODES.TXT*, *bootmagic*, *DEVCONF.TXT*, *FSCAN.CFG*, *MAGIC.MASTER*, and *SYSCONFIG.TXT*.

  - Manual page for the *allocator.config* file.

  - Manual pages for the supported */sbin* commands including the following: **create_pf**, **devstat**, **initpart**, **MAKEDEV**, and **top.**

  - Manual pages for supported Mach commands including the following: **machid**, **md**, **ms**, **vm_stat**, and **wh.**

  - Manual pages for **autoddb** and the load leveler.

  - Online manual pages now exist for the CLASSPACK Basic Math Library

- The *Paragon™ User's Guide* has been updated to include the following:

  - New "Improving Performance" chapter (chapter 8).

  - New "Using Pthreads" chapter (chapter 6).

  - Expanded "iPSC System Compatibility" appendix (appendix B).

  - New section on front panel LEDs (in chapter 1).

- The *Paragon™ System Administrator's Guide* has been updated as follows:

  - New "Configuring Partitions" chapter (chapter 7).

  - New "Paging Trees" chapter (chapter 10).

  - Rewritten "System Shutdown and Recovery" chapter (chapter 3).

- Rewritten "Managing PFS File Systems" chapter (chapter 9).
- New sections on creating device files for non-boot nodes and configuring additional RAID subsystems (chapter 8).
- Additional safety information and new section on marking slots as EMPTY (chapter 14).

PostScript Copies of the Manuals and Release Notes

PostScript copies of the Paragon manuals are available in the directory */usr/share/ps.docs* on the Paragon system.

PostScript copies of the *Paragon™ System Software Release 1.2 Release Notes* can be found in the directory */usr/share/release_notes*. An ASCII-only version of the list of system software bugs (*ss_buglist*) and list of bugs fixed since Release 1.0C (*ss_fixed*) can also be found in this directory. In the bug lists, the number on the left side of the first line for each bug is the bug number. Use this number when communicating with SSD Customer Support about the bug.

See the file */usr/share/README.docs* for more information about the contents of these directories.

Benchmark Programs

Benchmark programs used to obtain Paragon performance values are available for your use. Please contact the Parallel Systems Engineer or Field Applications Engineer associated with your site if you wish to access this software. Currently, a program measuring PFS performance is available.

Software Installation 2

Introduction

This chapter describes how to install Release 1.2 of the Paragon™ OSF/1 system software on the Paragon hardware.

If you have just received your Paragon supercomputer, the operating system, tools, and SATs are already installed. Skip ahead to "Configuring an Additional Ethernet Board" on page 2-35 and "Configuring the Paragon™ System for the Network" on page 2-32. If your system has I/O nodes in addition to the boot node, see "Configuring Additional RAID Subsystems" on page 2-38.

If you are upgrading to Release 1.2 from Release 1.1, do the complete installation beginning with "Information Needed for Installation" on page 2-3, and including "Configuring an Additional Ethernet Board" on page 2-35, and "Configuring the Paragon™ System for the Network" on page 2-32. If your system has I/O nodes in addition to the boot node, see "Configuring Additional RAID Subsystems" on page 2-38.

Reinstalling the system software or upgrading to Release 1.2 from one of the interim beta transmittals is similar to the complete installation, except that you may be able to skip some of the configuration steps. For example, the configuration of additional I/O nodes is preserved from the previous installation unless you are adding new hardware or choose to rebuild the root file system.

Before performing any installation, please contact your on-site Intel Parallel Systems Engineer. See the Preface section "Comments and Assistance" for information about contacting SSD Customer Support.

Typesetting and Other Conventions

The procedures in this chapter use the conventions described in the Preface. You should also be aware of the following conventions:

- The instruction "Enter ***character(s)***" means type the indicated character(s), and then press the **<Enter>** key. For example, "Enter ***y***" means type the letter 'y', and then press the **<Enter>** key.
- In prompts, square brackets surround a default value. Pressing **<Enter>** selects the indicated default value.
- Some steps in these procedures cause a great deal of information to be displayed. However, the steps as described here may show only the last message displayed. Also, do not be concerned if the indicated message does not appear immediately. Some steps take several minutes to complete.

Disk Space Requirements

Software: Paragon System Software Release 1.2
Installed on Paragon System hardware
Size: 95M bytes

Software: Paragon Boot Software Release 1.2
Installed on the diagnostic station
Size: 8M bytes

The boot software on the diagnostic station requires 7.5M bytes. However, using the diagnostic station to store tar files during installation requires additional temporary space of about 100M bytes

Information Needed for Installation

To perform the installation, you need to log in as *root* to the diagnostic station, so you will need the root password on the diagnostic station.

During the installation, you will be prompted for the following information:

Paragon host name (also called the network name):

Paragon Internet address:

Paragon disk type:
This is one of the following: RAID3, Maxtor76, Maxtor1240, 1gb, 4gbraid3.

Netmask address for your installation:

Broadcast address for your installation:

Gateway address for your installation:

Network time server for your installation:

You will also need network information about the system where you store the distribution files (the **tar** files on the distribution tapes). This system may be the diagnostic station.

Host name of the server with distribution files:

Internet address of the server with distribution files:

Path of the directory containing the distribution files:

Valid login name and password on the distribution server:

Powering Up the Paragon™ Hardware

The Paragon hardware must be powered up before you can install the operating system. Use the following procedure if the system is not already powered up.

1. Locate the circuit breakers at the bottom on the back of the each cabinet.

   If the circuit breakers are already up, there's no need to cycle them. Once the breakers are on, they should not be turned off.

   If the breakers are down (off), flip them up starting with cabinet 0 and proceeding in numerical order through the rest of the cabinets. Cabinet 0 is the cabinet containing the diagnostic station, and is the leftmost cabinet when viewed from the back or the rightmost when viewed from the front.

2. Open the back door of any cabinet. Notice the green power control board with red LEDs. This board has three white buttons on its upper-right corner.

   A. Press the bottom button to shut down power to the nodes.

   B. Wait about ten seconds for the disks to spin down and then press the top button to power up the nodes.

3. Power on the diagnostic station.

   A. The diagnostic station is located in cabinet 0, which is the rightmost cabinet when facing the cabinets from the front. Open the front door of this cabinet and swing out the diagnostic station.

   B. Turn on the power switch located on the back of the diagnostic station.

Logging into the Diagnostic Station

When the diagnostic station is powered up and in multiuser mode, log in as *root*. You can log in directly to the diagnostic station, or you can **rlogin** to the diagnostic station from another system on the network.

When you log directly into the diagnostic station, you have a choice to use either the Open Desk Top or the virtual terminal facility.

- **Using the Open Desk Top (ODT).** It is not necessary to run the Open Desk Top (ODT) on the diagnostic station. If you choose to do so, enter the following command:

  DS# ***startx***

  To create another window with the ODT, use the left mouse button to click on the root window and select "New Window" from the menu.

  The mouse has a trackball for cursor movement. The top button is considered to be the left button; the bottom button is the right. Use the mouse to switch between windows.

- **Using the virtual terminal facility.** If you choose to use the virtual terminal facility, use **<Alt-F1>** and **<Alt-F2>** to switch between windows.

<Alt-F1>	Puts you in the default window. This is the original window and is referred to as the diagnostic window.
<Alt-F2>	Switches to what will be referred to as the console window.

Verifying Required Hardware

Before performing any of the installation steps, verify the following information.

- Verify that there is an Ethernet connection from both the diagnostic station and from the Paragon boot node to your local network. This is needed to transfer the installation files.
- Verify that there is a serial line from the boot node to the diagnostic station. This line is needed for the Paragon console interface.
- If the MCP is disabled, the hardware for R1.1 will also support R1.2. To use the message coprocessor, node boards must be at the following revision levels:

317437-010	16M-byte GP node	(Fab 7, GP
317436-011	16M-byte MIO node	(Fab 7 MIO)
340267-001	16M-byte GP node	(Fab8/16 GP)
317421-007	32M-byte GP node	(Fab8/32 GP)
340130-006	32M-byte MIO node	(Fab 8 MIO)

In addition, the backplane must have the 0.1F capacitor added. Booting software automatically detects if the MCP hardware is installed and boots the appropriate software.

Verifying Configuration Files on the Diagnostic Station

The *MAGIC.MASTER*, *DEVCONF.TXT*, and *MAGIC.md* files must be in the boot directory (normally */usr/paragon/boot*) on the diagnostic station.

The SYSCONFIG.TXT file is automatically created during installation. An existing SYSCONFIG.TXT is saved as SYSCONFIG.OLD. Also, a *SYSCONFIG.BIN* file is created or overwritten in the diagnostic directory, */u/paragon/diag*. This is a hardcoded path inside **initutil**, and so *SYSCONFIG.BIN* should not be moved.

DEVCONF.TXT

The *DEVCONF.TXT* file that describes the configuration of the devices on the Paragon system. It contains one entry for each device in the system. This file must be in the boot directory (normally */usr/paragon/boot*).

The following is an example of a typical *DEVCONF.TXT* file:

```
DEVICES
ENET 00D03
TAPE 00D03 ID 5 DAT
TAPE 00D03 ID 6 DAT
TAPE 00C01 ID 6 DAT
RAID 00A01 ID 0 SW 3.04 LV 3 DC 5 SID 0 RAID3
RAID 00B01 ID 0 SW 3.04 LV 3 DC 5 SID 0 RAID3
RAID 00C01 ID 0 SW 3.04 LV 3 DC 5 SID 0 RAID3
RAID 00D03 ID 0 SW 3.04 LV 3 DC 5 SID 0 RAID3
RAID 01A12 ID 0 SW 3.04 LV 3 DC 5 SID 0 RAID3
RAID 01B12 ID 0 SW 3.04 LV 3 DC 5 SID 0 RAID3
MIO 00A01 H04
MIO 00B01 H04
MIO 00C01 H04
MIO 00D03 H04
MIO 01A12 H04
MIO 01B12 H04
HIPPI 00A05 H04
END_DEVICES
```

This file should already exist on your diagnostic station. Refer to the *Paragon™ XP/S Diagnostic Reference Manual* for more information about the format of this file.

MAGIC.MASTER

The **reset** script uses the *MAGIC.MASTER* file, which must contain the following lines:

```
BOOT_FIRST_NODE=boot_node
BOOT_CONSOLE=cm
```

In this listing, *boot_node* is an integer that uses the OS node numbering method to identify the boot node. The BOOT_CONSOLE string can be set to **f** to specify the **fscan** console interface or to **cm** to specify the **async** console interface using the serial port. **cm** is the default.

Enabling the Message Coprocessor

The bootmagic string BOOT_MSG_PROC specifies whether to boot the system with system software that supports the MCP hardware. The bootmagic string BOOT_MSG_PROC specifies the following:

0	Boot the system with the system software for MCP hardware disabled. The system will run with any hardware configuration, but the MCP functionality is turned off.
1	Boot the system with the system software that supports the MCP hardware enabled. If there is only MCP hardware in the system, the system boots and runs properly. If there is MCP and non-MCP hardware in the system, the system will not boot.

The booting software automatically sets the correct value of BOOT_MSG_PROC for the installed hardware. However, if you set BOOT_MSG_PROC to 0 in *MAGIC.MASTER*, you can force booting with the MCP disabled even if the hardware is present. Doing this, of course, lowers performance. If you set BOOT_MSG_PROC to 1 and do not have the needed hardware, the command **bootpp** (executed from within the **reset** script) displays the following message:

```
NOTICE FROM BOOTPP:
  The node found in Cabinet [x] BP [y] Slot [z] does not have the MCP ECO, but
  the bootmagic string BOOT_MSG_PROC is set to 1 in your MAGIC.MASTER file.
  You need to do one of the following:
    1. Replace the board with an ECO'd one OR
    2. Change your MAGIC.MASTER to contain BOOT_MSG_PROC=0 OR
    3. Remove the BOOT_MSG_PROC=1 entry from MAGIC.MASTER and the Message
       Copressor (MCP) will automatically be turned off during the
       boot proce
Bootpp Exiting...
```

The Parallel File System (PFS)

Set the following values in the *MAGIC.MASTER* file, depending on whether you use the Parallel FIle System (PFS). The default is for a non-PFS system.

If you do not have a PFS system, the following bootmagic variables need not be defined in *MAGIC.MASTER*. Their defaults are as follows:

```
NETIPC_PGLIST_HIGH=48
NETIPC_PGLIST_REFILL=40
NETIPC_PGLIST_LOW=32
```

If you have a PFS system, the following bootmagic variables must be defined in *MAGIC.MASTER*:

```
NETIPC_PGLIST_HIGH=103
NETIPC_PGLIST_REFILL=85
NETIPC_PGLIST_LOW=73
```

PFS makes large transient demands upon the microkernel when receiving multiple large messages from different nodes concurrently. The NETIPC_PGLIST_* bootmagic variables help prevent memory exhaustion when running some applications using PFS.

The NETIPC_PGLIST_* bootmagic variables permit tuning the microkernel's use of wired-down memory used for assembling inbound messages. Each unit represents one 8K page. The *_HIGH parameter indicates the maximum number of pages to dedicate to the pool. The *_LOW parameter indicates the minimum number of pages to dedicate to the pool. The *_REFILL parameter indicates the threshold at which resource depletion is detected and additional resources are allocated.

MAGIC.md

When you execute the **reset** script with the **ramdisk** option, it uses the *MAGIC.md* file instead of *MAGIC.MASTER*. *MAGIC.md* must contain at least the following lines:

```
BOOT_FIRST_NODE=0
REAL_FIRST_NODE=boot_node
BOOT_ROOT_DEV=md0a
NUM_BUFFERS=10
BOOT_VM_PAGESIZE=4096
```

Ensure that in this file, REAL_FIRST_NODE is equal to the BOOT_FIRST_NODE value in *MAGIC.MASTER*. As in the *MAGIC.MASTER* file, *MAGIC.md* uses the OS node numbering method. The exception is BOOT_FIRST_NODE in *MAGIC.md*, which does not follow a numbering scheme and must be 0.

NUM_BUFFERS=10 and BOOT_VM_PAGESIZE=4096 are the defaults and need not be explicitly set.

Backing Up

Before you begin the installation, you should back up any directories and files on your Paragon system that you wish to keep.

Normally you would back up files by copying them to a tape in the DAT drive or by packaging them into a compressed **tar** file and then using **ftp** to transfer the compressed file to a workstation on the network. For example, to back up */home* on a DAT tape, log into the Paragon system, insert a DAT tape into the drive, and issue the following commands:

```
# cd /home
# tar cvf /dev/io0/rmt6 *
```

The DAT device type is **rmt6** on the Paragon system, and the device type of the cartridge tape drive on the diagnostic station is **rStp0**.

Consider backing up the following files and directories on the Paragon system:

```
/etc/TIMEZONE
/etc/bootmesh.log  (for your records--do NOT restore on the new system)
/etc/devtab
/etc/fstab         (for comparison--do NOT restore on the new system)
/etc/pfstab        (for comparison--do NOT restore on the new system)
/etc/group
/etc/hippi.conf
/etc/hosts
/etc/hosts.equiv
/etc/networks
/etc/passwd
/etc/phones-file
/etc/printcap
/etc/profile
/etc/remote-file
/etc/resolv.conf
/etc/services
/etc/shells
/etc/uucp
/home              (user accounts)
/usr/local         (or whatever local directories you have)
/usr/spool/macs    (for comparison--do NOT restore on the new system)
/usr/spool/nqs     (for comparison--do NOT restore on the new system. The new version
                   of NQS uses different configuration rules)
```

```
/var/adm/cron/at.allow
/var/adm/cron/at.deny
/var/adm/cron/cron.allow
/var/adm/cron/cron.deny
/var/adm/sendmail/aliases
/var/adm/sendmail/sendmail.cf
/var/adm/wtmp          (for your records--do NOT restore on the new system)
/var/spool/mail
```

Also, you might want to save the output from a **last** command for your history records.

Installing the Software

- **Installation Time**. Approximately 2 hours for the complete update from Release 1.1 to Release 1.2.

- **Installation Medium**. The system software is provided on two 0.25-inch QIC 150 cartridge tapes. In some special cases, installation **tar** files are transferred from SSD via **ftp**

  - Paragon™ System Software Release 1.2 (313095-001)

boot.tar	Contains all the utilities for booting Paragon OSF/1 from the diagnostic station as well as Paragon OSF/1 diagnostic binaries and debugging tools.
root.tar	Contains all the commands that are installed on the root partition.
usr.tar	Contains all the commands that are installed on the */usr* partition.
mach_svr.tar	Contains the microkernel, initialization, and OSF/1 server files.
doc.tar.Z	Contains online manual pages and PostScript copies of manuals for the operating system.

  - Paragon™ SAT Software Release 1.2 (313097-001)

sat.tar.Z	Contains the System Acceptance Tests source and binaries.
sat.doc.tar	Contains a PostScript copy of the *Paragon™ XP/S System Acceptance Test User's Guide.*

NOTE

These instructions assume that SCO UNIX and the SCO Open Desktop are installed on the diagnostic station. If this software is not installed, contact Intel SSD Customer Support.

Obtaining the Installation Files from Tape

If you have the installation files on one or more tapes, copy the files on each tape to the diagnostic station. These files may go into the */tmp* directory or the */u/tmp* directory. The following directions assume */tmp*. You may choose instead to copy the **tar** files to one of your local servers.

1. Change to the directory where the distribution files will be stored:

   ```
   DS# cd /tmp
   ```

2. Insert the distribution tape into the cartridge tape drive.

3. Use the **tar** command to copy the files from the distribution tape.

   ```
   DS# tar xvfp /dev/rStp0
   ```

   After the files have been copied, remove the tape from the cartridge tape drive.

Installing Boot Files on the Diagnostic Station

The following instructions assume that you are logged in as *root* to the diagnostic station.

1. If you copied the distribution files from a tape into the */tmp* directory on the diagnostic station, proceed to the next step.

   If the distribution files are stored on some other server, you need to use **ftp** to copy the *boot.tar* file to */tmp* or */u/tmp*. This file requires about 8M bytes of storage. When you invoke **ftp**, it will prompt you for a login name and password on the distribution server.

   ```
   DS# cd /tmp
   DS# ftp IP Address of server with distribution files
   Name: login_name
   Password:password
   ftp> cd path to distribution files
   ftp> bin
   ftp> get boot.tar
   ftp> bye
   ```

2. Change to the / directory, and use **tar** to extract the files from *boot.tar*:

CAUTION

Ensure that no one is using any of the utilities from */usr/local/bin* when you untar *boot.tar*. If someone is using a utility when you untar *boot.tar*, that utility will not be updated.

CAUTION

Many sites backup boot software by moving directories to another name. If you do this, untarring *boot.tar* will create directories. Unless you set your umask appropriately, the permissions on these directories will be wrong, and the Paragon system will not boot.

```
DS# cd /
DS# umask 022
DS# tar xvfp /tmp/boot.tar
```

3. If you are upgrading from Release 1.1, you must relink the kernel on the diagnostic station. To relink the kernel, issue the following commands:

```
DS# cd /etc/conf/pack.d/scan
DS# ./buildscan
```

This command takes a few minutes to rebuild the UNIX Operating System. Answer **y** to the questions. The first question asks if you want the new kernel to boot by default. The second question asks if you want the kernel environment rebuilt. Then, when the DS# prompt returns, reboot.

```
DS# init 6
```

- If you are working directly at the diagnostic station, wait for the : (colon) prompt, and then press the **<Enter>** key.

- If you are remotely logged into the diagnostic station, the **init 6** closes the connection and you will need to **rlogin** again after waiting a few minutes. You can use **ping** to detect if the diagnostic station is ready to be logged into.

When the login prompt appears, log in as *root*.

4. Ensure that the following variables are defined correctly in */etc/default/tcp*:

```
DOMAIN=
NETMASK=
BROADCAST=
```

5. Verify that the */etc/hosts* file contains the two aliases: *PARA_ALIAS* and *DIAG_ALIAS*. This modification needs to be performed only once during the first installation.

   - The *PARA_ALIAS* alias identifies the IP address of the Paragon system.

- The *DIAG_ALIAS* alias identifies the IP address of the machine where the kernel, ramdisk, and *bootmagic* files exist. This machine is usually the diagnostic station.

Here is an example of the two lines as they might appear in */etc/hosts.*

```
xxx.yy.zzz.a  ds ds.abc.def.com DIAG_ALIAS  #diagnostic station
xxx.yy.zzz.ab paragon           PARA_ALIAS  #Paragon system
```

6. If the diagnostic station's and the Paragon system's IP address and system network name are not what you want, change the diagnostic station's IP address and system network name in the file */etc/hosts* to the values for your installation. You should have values for the diagnostic station, the Paragon system, and loopback.

A. Issue the **uname** command, specifying the new system name.

```
DS# uname -S <new system name>
```

B. Change to the directory */etc/conf/cf.d* and run **configure**.

```
DS# cd /etc/conf/cf.d
DS# ./configure
```

C. A menu appears. Choose System Name and type in the new system name.

D. Link in a new kernel. This step is only necessary if you changed the system name. Your current directory is still */etc/conf/cf.d.*

```
DS# ./link_unix
```

E. Reboot the diagnostic station. Answer **y** to all questions. There's a silent wait here of a few minutes. Press any key when requested.

```
DS# init 6
```

Installing the Paragon™ Software

1. Ensure that the DEVCONF.TXT file in */usr/paragon/boot* is present and correct. Refer to the *Paragon™ Commands Reference Manual* for information about how to set up this file.

2. Reset the Paragon system using the **reset** command with the **autocfg** option. The **reset** script is located in */usr/paragon/boot*. For information about the **reset** command, refer to the *Paragon™ Commands Reference Manual.*

   When executed with the **autocfg** option, the **reset** script creates the file *SYSCONFIG.TXT* in the current directory and *SYSCONFIG.BIN* in the diagnostic directory */u/paragon/diag*. A previous version of SYSCONFIG.TXT is saved as SYSCONFIG.OLD.

   This script resets the nodes, resets the mesh, waits for the Node Confidence Tests (NCT) to complete, downloads bootmagic strings, and initiates booting. By default, it uses the serial line with the **async** utility. To exit **async** and return to the DS# prompt, enter the key sequence ~. (or ~~. if you used **rlogin** to log into the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins.

```
DS# cd /usr/paragon/boot
DS# ./reset autocfg
    .
    .
    .
---> Automatic SYSCONFIG.TXT generation.
How many cabinets are there? [1] number of cabinets
Generating /u/paragon/diag/hwconfig.txt...
Generating SYSCONFIG.TXT...
SYSCONFIG.TXT has been created.
Generating SYSCONFIG.BIN...
Using MAGIC.MASTER as the Magic file.
Creating bootmagic file...
Using 'fscan' as the console interface tool...
DS#
```

3. Reset the Paragon system using the **reset** script with the **ramdisk** option.

   The <ramdisk> prompt appears when the Paragon RAM disk has been loaded.

```
DS# ./reset ramdisk
    .
    .
    .
INT: SINGLE-USER MODE
<ramdisk>
```

4. Execute the **install** script.

   ```
   <ramdisk> install
   ```

   This script asks you to verify or provide the information necessary to complete the installation. The script gets default values from existing configuration files, as well as from information gathered during previous installations. You should be prepared to provide the following information:

   Host name of the Paragon system
   IP address of the Paragon system
   Disk type of the Paragon system

   Netmask address of the Paragon system
   Broadcast address of the Paragon system
   Gateway address of the Paragon system

   Host name of the distribution system
   IP address of the distribution system
   Pathname to the directory containing the installation **tar** files on the distribution system
   Login name and password on the distribution system

5. The **install** script asks for the type and SCSI ID of the disk connected to the boot node on the Paragon system, and then lists the five supported disk types. Press the **<Enter>** key to confirm the default value shown in square brackets (for example, [raid3]), or enter the correct value. For the disk type, you can enter the disk number (for example, '0' for raid3), or you can spell out the name of the disk type (for example, "raid3"). If you specify an invalid number or name, the script detects the inconsistency with the disklabel and prompts you again for a choice, listing acceptable choices.

   Note that the script detects no difference between RAID3 and RAID5. If you have a RAID on your system, it will be set up as RAID3. If the drive is currently set up for RAID5, specify "raid3" when prompted for disk type, and it will be automatically converted from RAID5 to RAID3.

   The SCSI ID is a number between 0 and 6 and identifies the address of the disk on the SCSI controller. This number is usually '0' for a Paragon system with more than two backplanes.

```
=============================================
Paragon Operating System Installation
O.S. Install
=============================================
```

```
------------------------------------------------
0) raid3                    4.70 Gigabyte Raid
1) maxtor76                 0.76 Gigabyte Maxtor
2) maxtor1240 (non-raid)  1.24 Gigabyte Maxtor
3) 1gb (non-raid)           1 Gigabyte Drive
4) 4gbraid3                 4 Gigabyte Raid Drive
------------------------------------------------
Disk Type: [raid3]
SCSI ID for Boot Disk: [0]
```

The script then displays the information you have entered, and asks you to confirm it. If the information is correct, press the **<Enter>** key, or enter '**Y**' or '**y**'. These disk information prompts repeat until you confirm that the correct information has been entered.

```
============== DISK INFORMATION SUMMARY ===================
Disk Type is "raid3" (Disk Label will be "raid3")
SCSI ID 0

Is this correct? (y/n)[y]
```

If the disk is a RAID, the **install** script checks the configuration and performs the following actions:

- If the RAID controller is not an NCR, the script generates an error message and exits.
- If the status of the RAID controller is anything other than "Optimal" or "Degraded", or if the RAID level is anything other than '3', the script displays the following messages:

```
WARNING: The RAID drives are either configured as RAID5,
or are otherwise in need of initialization. This procedure
will destroy any data currently on the drive. Do you want
to continue? (y/n)[n]
```

To proceed with the installation, you must enter '**Y**' or '**y**' to both questions, and the drive is initialized. If you enter anything else, the **install** script exits.

6. The **install** script attempts to mount the root file system.

A. If the root file system exists, the **install** script executes the **fsck** utility to clean it (if necessary). Even though the root file system exists, you may want to rebuild it to create a clean file system with no old files on it. The script displays the following prompt to ask if you want to rebuild the file system:

```
A root file system has been detected on /dev/rz0a.
It may not be necessary to rebuild it. If you do rebuild it,
all files in the existing root file system will be destroyed
Do you want to rebuild it? (y/n)[n]
```

If you press the **<Enter>** key, or enter '**N**' or '**n**', the installation proceeds without rebuilding the file system. If you enter '**Y**' or '**y**', the script asks for confirmation:

```
Creating New Root File System. Okay? (y/n)[n]
```

At this point, you must enter '**Y**' or '**y**' to rebuild the file system and proceed with the installation. Any other response terminates the installation.

B. If the root file system does not exist, the script prompts for permission to create a new root file system. To proceed with the installation, you must enter '**Y**' or '**y**'. If you enter anything else, the **install** script exits.

```
Creating New Root File System. Okay? (y/n)[n] y
```

7. After the root file system has been mounted, the **install** script verifies the existence of the */usr*, */etc*, */etc/defaults*, and */pfs* directories, and creates them if they do not exist.

   If the */usr* and */pfs* file systems exist, you may want to rebuild either or both of them to create a clean file system with no old files on it. For each of these file systems, the **install** script displays a message asking if you want to rebuild an existing file system. A typical message is as follows:

```
A pfs file system has been detected on /dev/rz0d.
It may not be necessary to rebuild it. If you do rebuild it,
all files in the existing pfs file system will be destroyed
Do you want to rebuild it? (y/n)[n]
```

   If you press the **<Enter>** key, or enter '**N**' or '**n**', the installation proceeds without rebuilding the file system. If you enter '**Y**' or '**y**', the script rebuilds the file system and proceeds with the installation.

8. A new "paging" file system will be created. You do not have a choice in this. The following block and fragment sizes will be used:

   /paging Block=8K Fragment=2K

9. Next, the **install** script asks you to verify the information needed to get the distribution files. If a */etc/defaults/install* file was saved from a previous installation, the **install** script summarizes the file's information and asks if it is correct. If */etc/defaults/install* does not exist or has information missing, the script asks the needed questions. The following listing shows an example of the summary:

```
================= RESPONSE SUMMARY ================
Node Name:          system
IP Address:         00.00.00.00
Gateway:            133.26.1.1
Netmask:            255.255.00.00
Broadcast:          133.26.255.255
Distrib Node:       para_ds
Distrib IP Addr:    133.26.15.5
Distrib Path:       /tmp
Create /home:       Y
---------------------------------------------------
Is the above information correct? (y/n)[n]
```

If you enter '**Y**' or '**y**', the installation proceeds using the default values. If you enter anything else, or if the */etc/defaults/install* file was not found, the script displays a series of prompts for you to enter the information. The default value for each item is displayed in square brackets. Press the **<Enter>** key to select the default, or enter a new value. In the following example, the installer entered new values for "System Name" and "System IP Address", and accepted the default values for the other items.

```
System Name [system]: paragon512
System IP Address [00.00.00.00]: 133.26.101.31
Gateway [133.26.1.1]:
Netmask [255.255.00.00]:
Broadcast Address [133.26.255.255]:
Distribution Node Name [para_ds]:
Distribution IP Address [133.26.15.5]:
Distribution Path [/tmp]:
Create /home File System [Y]:        N
```

At the end of the questions, the script displays the new information in the response summary and asks again for confirmation. This sequence is repeated until you indicate that the response summary is correct.

10. The **install** script sets up the network using the values you just provided. It then asks for a login name for **ftp** to use when getting files from the distribution system. The distribution system is the system from which you obtain the installation **tar** files. This may be the diagnostic station or one of your local servers.

```
Username for FTP'ing files from para_ds:[anonymous]
```

Enter a valid user name. The script then invokes **ftp** and attempts to use the name to log into the distribution node. When **ftp** prompts you, enter the user name's password.

If the login is unsuccessful, the script displays an error message along with a request to verify the distribution information.

```
ERROR: [file] was not retrieved.

=========== DISTRIBUTION INFORMATION ==============
Distrib Node:      para_ds
Distrib IP Addr:   133.26.15.5
Distrib Path:      /tmp
----------------------------------------------------
Is the above information correct? (y/n) [y]
```

Entering '**n**' gives you the chance to change this information. If the information is correct, enter '**y**' and the script repeats the process of asking for a login name and password, and using **ftp** to get the distribution files.

If the login is successful, **ftp** transfers the *root.tar*, *mach_svr.tar*, *hosts*, and *usr.tar* files from the distribution system

11. The remainder of the Paragon installation is automatic, requires about 30 minutes to complete, and performs the following:

    - Preserves copies of */etc/hosts*, */etc/passwd*, */etc/exports*, */etc/group*, */etc/fstab*, */etc/printcap*, */etc/pfstab*, */etc/devtab*, */etc/services*, */etc/resolv.conf*, */etc/ntp.conf*, */etc/shells*, */var/adm/sendmail/sendmail.cf*, */var/adm/sendmail/sendmail.cw*, and */var/adm/sendmail/sendmail.st* if they exist.

    - Restores the **tar** files that were retrieved from the distribution node.

    - Creates the default */etc/fstab*, */etc/devtab*, and */etc/rc.config* files.

    - Restores copies of */etc/hosts*, */etc/passwd*, and */etc/exports*, */etc/group*, */etc/fstab*, */etc/printcap*, */etc/pfstab*, and */etc/devtab* if they exist.

    - Makes PFS directories.

    - Makes the paging file. For RAID drives, the script creates a 64M-byte paging file. For other drives, it creates a 32M-byte paging file. It may take up to 10 minutes to make a 64M-byte paging file.

    When the <ramdisk> prompt returns, the installation is complete.

12. By default, the **install** script creates a 64M-byte paging file for the default pager on the boot node. A better size is 512M bytes. If you do not increase the size of the paging file now, you can still increase it after the installation.

The following commands show how to increase the size of the paging file to 512M bytes. The new paging file is placed in the */home* partition because the root partition is not large enough for such a file. (The root partition is / on the RAID subsystem and */root* on the ramdisk.)

NOTE

Increasing the page size to 512M bytes can take up to thirty minutes. A smaller page size will take less time. Before you increase page size, you should make sure there is enough room in the home partition.

```
<ramdisk> mount -u /
<ramdisk> fsck -y /dev/rrz0a
<ramdisk> mount -w /dev/rz0a /root
<ramdisk> fsck -y /dev/rrz0f
<ramdisk> mount -w /dev/rz0f /home
<ramdisk> cd /home
<ramdisk> /root/sbin/create_pf 512M paging_file
<ramdisk> chmod 600 paging_file
<ramdisk> cd /root/dev
<ramdisk> ln io0/rz0f rz0f
<ramdisk> cd /root/mach_servers
<ramdisk> mv paging_file paging_file.orig
<ramdisk> /root/sbin/ln -s /dev/rz0f/paging_file paging_file
<ramdisk> cd /
<ramdisk> sync
<ramdisk> umount /root
<ramdisk> umount /home
```

edit rel notes 1.2

renew FS

13. When the <ramdisk> prompt appears, return to the diagnostic station prompt and use the **reset** command to reboot the Paragon system. Disconnect from the Paragon RAM disk by typing **~.** (tilde period) or **~~.** if you are logged in remotely to the diagnostic station. The key sequence **~q** also works and does not require that you keep track of the number of remote logins.Then, invoke the **reset** command.

```
<ramdisk> ~.
Exiting ...
DS# ./reset
    .
    .
    .
# <Ctrl-D>
```

When the **reset** command has completed, the prompt that appears is on the Paragon system. Enter multiuser mode by pressing **`<Ctrl-D>`**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1. The root file system is checked and mounted and then the mesh is booted.

This step automatically brings up the remaining nodes on the mesh, using the */sbin/bootmesh.sh* script, which is called by */sbin/bcheckrc*. The mesh will be booted just after the root file system is checked and mounted. Any nodes which fail to boot will be reported to the console; you can also look in the file */etc/bootmesh.log* for the list of failed nodes.

If for some reason you need to boot with mesh booting disabled, add the string

```
DISABLE_BOOTMESH=1
```

to */usr/paragon/boot/MAGIC.MASTER* on the diagnostic station. If you have booted with this string, you can later boot "by hand" by creating the Paragon file */etc/.forcebootmesh*, and executing */sbin/bootmesh.sh*.

When the system completes booting to multiuser mode, you can login again. If you have increased the size of the paging file and successfully booted to multiuser mode, you may delete the original paging file, */mach_servers/paging_file.orig*.

Executing the *postboot* Installation Script

When the installation is complete and the Paragon system is in multi user mode, run **postboot** for the final configuration. **postboot** updates the message of the day, sets the date/time, and optionally configures the network time daemon, prompts for compiler/documentation and tools installation, and creates the terminfo databases, spell dictionaries, cat directories, and lint libraries.

If you intend to use postboot to install the compilers, you must have already read in the installation files from the compiler tapes. Refer to "Obtaining the Installation Files from Tape" on page 2-14 for instructions on how to read the distribution tape.

To execute the **postboot** script, type the following:

```
# cd /
# ./postboot
```

The script asks you to confirm the current date and time, and then prompts you before performing each of several tasks.

```
================================================================
                     Postboot Process

This script will perform final configuration of your Paragon system.


Updating message of the day (motd) ...

Is this correct date/time:  Wed Apr 27 10:30:36 PST 1994  (y/n)? [y]
n

Enter date as yymmddhhmm
9404270913
Wed Apr 27 09:13:00 PDT 1994

The remaining tasks this script will perform are:

  - Configure Network Time Daemon
  - Install compilers and/or documentation
  - Install Paragon tools
  - Create the terminfo databases
  - Create the spell dictionaries
  - Create the cat directories
  - Create the lint libraries
      Note: You must install the C compiler before
      the lint libraries can be created.

Before each task is performed you will have the chance to
```

```
choose if you want the task performed.
-----------------------------------------------------------
```

The first task is to specify a network time daemon. Be prepared to provide the name of the computer that maintains your network-wide clock.

```
Configure Network Time Daemon ...

Do you wish to proceed with configuring network time daemon (y/n)?
[y]
```
n

If you want to configuring the network time daemon, enter *y*. Note that the network time daemon adversely impacts performance because it sends messages to all the nodes in the system every five seconds. If you choose *y*, you get the message,

```
What is name of your network time server?
```
name of network time daemon

A warning appears if you have already defined a network time daemon. If a warning appears and you want to change the definition, confirm that you want the daemon configured.

```
Setting up network configuration file '/etc/ntp.conf'
Starting the network time server ...
Network Time Service started
-----------------------------------------------------------

Installing compilers and/or documentation...

Do you wish to proceed with installing compilers and/or
documentation (y/n)? [y]
```
y

If you want to skip installing the native compilers, enter ***n***. If you answer *y*, the following menu is displayed.

```
================= RESPONSE SUMMARY ===============================
Install C compiler and its docs:             Y
Install FORTRAN compiler and its docs:       Y
Install system software documentation:       Y
Distribution Node:                           Distribution Node Name
Distribution IP Addr:                        Distribution IP Addr
Distribution Path:                           Distribution Path
------------------------------------------------------------------
Is the above information correct? (y/n)[n] y
```

Answer ***n*** to change the defaults.

```
Installing Paragon tools...

Do you wish to proceed with installing Paragon tools (y/n)? [y]y
```

If you want to skip installing the tools, enter ***n***. If you answer *y*, the following menu is displayed

```
================= RESPONSE SUMMARY ==============================
Install standard Paragon tools:          Y
Install motif:                           Y
Install dgl:                             Y
Install opengl:                          Y
Distribution IP Addr:                    Distribution IP Addr
Distribution Path:                       Distribution Path
-----------------------------------------------------------------
Is the above information correct? (y/n)[n] n
```

Answer ***n*** to change the defaults. If you choose to install the tools, appropriate online documentation is also installed.

```
Creating terminfo databases....

Do you wish to proceed with creating terminfo databases (y/n)? [y]y
Running terminfo translator....
----------------------------------------------------------

Creating spell dictionaries....

Do you wish to proceed with creating spell dictionaries (y/n)? [y] y
----------------------------------------------------------

Creating cat directories....

Do you wish to proceed with creating the cat directories (y/n)? [y] n
Cat directories will not be created.
----------------------------------------------------------
```

If you choose to create the cat directories, note that this choice also creates the cat files. This choice takes a long time to execute and uses a great deal of disk space.

The cat files are formatted versions of the online manual pages. Because it is quicker to access a formatted manual page than an unformatted one in */usr/share/man*, some users choose to store cat files. However, if the cat directories exist and are empty, the first access to an online manual page places a formatted version in the appropriate cat directory. Hence, making empty cat directories results in, after a while, a collection of formatted files for the most frequently accessed online manual pages. This is preferable to making formatted versions of all online manual pages.

If you choose to create the *cat* directories, you might see errors similar to the following, which can be ignored: `unterminated list (no .LE) -- noticed by the .SH "RELATED".`

```
Creating lint libraries....

Do you wish to proceed with creating the lint libraries (y/n)? [y] y
```

NOTE

To install C and Fortran cross compilers, refer to the *Paragon™ C Compiler Release 4.5 Software Product Release Notes* and the *Paragon™ C Compiler Release 4.5 Software Product Release Notes.*

NOTE

After **postboot** has successfully run, consider deleting the installation tar files from the distribution system. They will only be needed if you want to reinstall the Paragon OSF/1 operating system.

If you have installed any online documentation, you must make the *whatis* database. This is the database used by the **man** command with the **-k** option. To make the *whatis* database, issue the following commands:

```
# cd /usr/share/man
# /usr/lbin/mkwhatis
```

mkwhatis is does not successfully make the complete *whatis* file. It returns an error message that says "argument too long." The same error message is returned by **catman -w**.

Configuring the Allocator

initpart is run when the Paragon system enter multiuser mode. **initpart** creates a new */etc/nx/.partinfo* file if the current *.partinfo* has xy dimensions that are inconsistent with reality or the file does not exist.

initpart also creates the default configuration file for the allocator. This file sets the user model to allow for one gang-scheduled partition with a maximum of two NX applications per node and a minimum rollin quantum of 1 hour. The default *allocator.config* file looks as follows:

```
NUM_GANG_PARTS=1
DEGREE_OF_OVERLAP=2
MIN_RQ_ALLOWED=1h
```

The value of `MIN_RQ_ALLOWED` must be greater than zero. You can disable any of these values by removing the appropriate variable from */etc/nx/allocator.config*.

If you remove the file, it will be created with defaults at the next boot. If you want to remove constraints on the user model, comment out the appropriate lines. Indicate a comment by putting a # in the first position of the line.

NOTE

Beginning with Release 1.2, you should not use the **-tile** and **-MACS** options on the *allocator* line in */sbin/init.d/allocator.* Use the *allocator.config* file instead. Refer to the *allocator.config* manual page in the *Paragon™ Commands Reference Manual.*

To have any changes to the configuration file take effect you will need to stop and restart the allocator after the changes are made. To stop and start the allocator, issue the **allocator** command.

```
# /sbin/init.d/allocator stop
# /sbin/init.d/allocator start
```

initpart now makes the root partition with permissions 754 and sets the scheduling attribute to SPS, or space share. This will allow only *root* to create subpartitions. So when you are installing the system you need to take into account who you want to make partitions and where they can make them.

If you do not specify a scheduling characteristic when making a partition (for example, **-sps** for space sharing) or use **-rq**, then the characteristics will be inherited from the parent, which in this case is space-shared. To explicitly set the *.compute* partition to be gang-scheduled, you must specify a rollin quantum value, (0 is valid, and means an application will run to completion). Here is an example of creating a two-node service partition (assuming node 2 is a service node and node 3 is the boot node) and a fourteen-node, gang-scheduled compute partition:

```
# mkpart -nd 2,3 -ss .service
# mkpart -sz 14 -rq 0 -mod 777 .compute
```

To leave the compute partition as a space shared partition you would issue:

```
# mkpart -sz 14 -mod 777 .compute
```

Creating the *service* and *compute* Partitions

Use the **mkpart** command to create the *service* and *compute* partitions. The following example assumes a 64 node system with the boot node at node 7 and additional MIO nodes at nodes 15 and 23. These are OS node numbers.

CAUTION

The *.service* partition must contain the boot node.

Create the *service* partition. Use the **-ss** switch, so the service partition has standard scheduling. The following example allocates nodes 7, 15, and 23 to the service partition.

```
# mkpart -nd 7,15,23 -ss .service
```

Create the *compute* partition using the **-sz** switch to allocate the remaining 61 nodes to the compute partition and the **-mod** switch to set permissions that enable all users to create subpartitions in the compute partition. Note that you may want to change the permissions of the compute partition when you personalize your configuration.

```
# mkpart -sz 61 -mod 777 .compute
```

After creating the compute partition, issue the **showpart** command to display the nodes that are allocated to each partition. Verify that the partitions do not overlap (that is, each node is allocated to only one partition). For example:

```
# showpart .compute
# showpart .service
```

If the partitions do overlap, use the **rmpart** command to remove the overlapping partitions. When you use **mkpart** to create the partitions again, you may need to use the *-nd* switch to specify the node numbers of the nodes to include in the *compute* partition. For example:

```
# rmpart .compute
# mkpart -nd 0..5,8..14,16..22,24..63 -mod 777 .compute
```

Specifying Your Time Zone

If your Paragon system arrived with the software already installed, you might need to modify the */etc/TIMEZONE* and */etc/rc.config* files on the Paragon system to reflect your time zone. You may also need to modify these files if you have reinstalled the Paragon software.

Configuring the Paragon™ System for the Network

If your Paragon system arrived with the software already installed, you must change the network name and IP address of the diagnostic station and the Paragon system to bring them up on the network at your site. If you are reinstalling or upgrading the system software, you should only need to reconfigure the Paragon system.

If your Paragon system has any HIPPI boards installed, you must configure their network interface when you first receive the system, and also whenever the system software is installed or upgraded. HIPPI configuration includes activating the network interface and specifying the HIPPI routing tables. For a complete description of this procedure, see the *Paragon™ XP/S High-Performance Parallel Interface Manual.*

1. Edit the */etc/rc.config* file, and change HOSTNAME and HOSTID (IP address) to the new values.

2. Verify that the */etc/hosts* file has entries for the Paragon network ID (IP number) and diagnostic station network ID.

3. If you are using the domain nameserver, edit */etc/resolv.conf*. Set the value for the domain and add nameservers if needed.

Resetting the System After Installation

Resetting the Paragon system, once it is up and running, is performed as follows:

1. Log in or **rlogin** to the diagnostic station as *root.*

2. Connect to the Paragon system with the **console** command. The **console** command is a script that the **reset** script created in */usr/paragon/boot* the last time **reset** was run. **console** uses whatever the console connection was at the time and supports the **async**, **fscan**, and the **scanio** console interfaces. You may have to press a carriage return after issuing **console** to get a prompt.

```
DS# cd /usr/paragon/boot
DS# ./console
          .
          .
login:
```

3. Login as *root* to the Paragon system, unmount all file systems, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins. Then, use the **reset** script to reboot the system.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS# ./reset
          .
          .
# <Ctrl-D>
```

When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1 .

When you boot the system, the **fsck** command is automatically run on each file systems defined in */etc/fstab* (such as */home*, */pfs* and the PFS stripe directories typically mounted on */home/.sdirs/vol**).

In the case of a severely damaged file system, the **fsck** will fail and a message will be displayed to the screen. The message will scroll off the screen before the reboot is complete and you may not notice it. The system will continue to boot to a multi-user state with no other indication that anything is wrong.

If the **fsck** fails, the affected file system will not be mounted. Any files written to the unmounted file system are written into the root file system, not the desired file system. Later, if you notice that a file system did not mount and you **fsck** and mount it, the files that were written into the root file system become hidden. To avoid this problem, after each boot you should manually compare the mounted file systems with the contents of the *fstab* file. Make sure that the PFS stripe directories are mounted or the **fsck** data will go into the */home* file system.

For example, assume that **fsck** failed when processing the PFS stripe directories and that these stripe directories are mounted on the partition */dev/io1/rrz0c*. To manually run **fsck** on this partition, enter

```
# fsck -y /dev/io1/rrz0c
```

The **-y** directs **fsck** to answer yes to all its questions.

User Notification

Consider adding the following messages to */etc/motd*:

```
Release 1.2 of the Paragon system software has
been installed on this system. All user
applications must be recompiled and relinked
before they can be run.

PostScript copies of the release notes for
Release 1.2 of the Paragon system software can be
found in the directory /usr/share/release_notes.
This directory also contains ASCII and PostScript
copies of the current list of system software
bugs.
```

Configuring an Additional Ethernet Board

Every Paragon system has at least two Ethernet connections, one to the diagnostic station and one to the boot node. Follow these instructions if you have another Ethernet connection to an I/O node other than the boot node. This makes for a total of three Ethernet connections, but because this is a second Ethernet connection to the Paragon nodes, these instructions are often referred to as those for configuring a second Ethernet.

1. Log in or **rlogin** to the diagnostic station as *root*.

2. Change directory to */usr/paragon/boot*. Edit *DEVCONF.TXT* to contain a line identifying the position of the second Ethernet connection.

   For example, consider a two-cabinet configuration with one Ethernet connection. This connection is on the boot node which is in cabinet 0, backplane D and slot 3. The CBS number of the boot node is 00D03. Now assume that you add a second Ethernet connection to a new I/O node in cabinet 1, backplane A, and slot 12. Its CBS number is 01A12.

   A typical *DEVCONF.TXT* file is shown below. The lines in **bold** are those added for the new I/O node with the Ethernet connection.

```
DEVICES
ENET 00D03
ENET 01A12
RAID 00D03 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3
TAPE 00D03 ID 6 DAT
MIO 00D03 H04
MIO 01A12 H04
END_DEVICES
```

3. Connect to the Paragon system with the **console** command.

```
DS# cd /usr/paragon/boot
DS# ./console
        .
        .
        .
login:
```

4. Edit the /etc/hosts file to include the new interface, using the additional IP address and additional host name.

5. Edit the file */etc/rc.config*. Define *HOSTNAME2, HOSTID2* and *NETDEV2* and export their values.

```
HOSTNAME=joe
HOSTNAME2=frank
HOSTID=xxx.yy.zz.aaa
HOSTID2=xxx.yy.zz.bbb
NETMASK=255.255.255.00
NETDEV="<7>em0"
NETDEV2="<96>em0"
        .
        .
        .
export DISPLAYTYPE HOSTNAME HOSTNAME2 HOSTID HOSTID2 NETMASK
NETDEV NETDEV2 TZ
```

The NETDEV variable identifies the node number of the Ethernet connection. Each Ethernet connection has its own HOSTNAME and HOSTID. It is not necessary to define a NETMASK2 if the second Ethernet connection is on the same class as the primary and uses the same mask value.

6. Login as *root* to the Paragon system and edit the file */sbin/init.d/inet*. The lines in **bold** indicate the lines that must be added to configure a second Ethernet board. Both are **ifconfig** lines.

```
if [ "$ARGTWO" = "force" -o "$9" = "S" ]; then
 echo "Configuring network"
 echo "hostname: \c"
 /sbin/hostname $HOSTNAME
 /sbin/hostid $HOSTID
 echo "hostid: $HOSTID"
 /sbin/ifconfig $NETDEV $HOSTID netmask $NETMASK -trailers
# ifconfig second ethernet
 /sbin/ifconfig $NETDEV2 $HOSTID2 netmask $NETMASK -trailers
 /sbin/ifconfig lo0 127.0.0.1
#
# Add default routes; start routed
#
#                echo "Adding route for osfgw: "
#                SUBNET=`expr $HOSTID : ".*`.*`<.*>`.*"`
#               /sbin/route add default 137.27.$SUBNET.1
                /sbin/route add default $GATEWAY
fi
;;
'stop')
        /sbin/ifconfig $NETDEV2 down
        /sbin/ifconfig $NETDEV down
        /sbin/ifconfig lo0 down
```

7. Unmount all file systems listed in */etc/fstab*, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins. Then, issue **reset** with the **autocfg** option. This ensures that the new information in *DEVCONF.TXT* is added to *SYSCONFIG.TXT*.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS# ./reset autocfg
DS#
```

8. Reset the Paragon system and enter multiuser mode. When the **reset** command has completed and the prompt returns, enter multiuser mode by **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1 .

```
DS# ./reset
        .
        .
        .
# <Ctrl-D>
```

Configuring Additional RAID Subsystems

This section describes the configuration steps that are necessary if your Paragon has I/O nodes in addition to the boot node that have a RAID subsystem attached.

1. Login to the diagnostic station as *root.* Determine the CBS (Cabinet:Backplane:Slot) numbers for your I/O nodes, and then check the *SYSCONFIG.TXT* file in the directory */usr/paragon/boot* for correctness.

   For example, consider a configuration that has three I/O nodes, one of which is the boot node. The I/O nodes are located as follows.

   - The boot node has an OS number of 7 and is located in cabinet 0, backplane D, and slot 3. Its CBS number is 00D03.

   - The second MIO node as an OS number of 15 and a CBS number of 00D02. This is an additional I/O node.

   - The third MIO node has an OS number of 23 and a CBS number of 00D01. This is an additional I/O node.

   The appropriate portion of a correct *SYSCONFIG.TXT* for this example is shown below. Note that the lines for slots 1, 2, and 3 contain the keyword RAID3 and MIO. Also note that slot 3, which contains the boot node, also has an Ethernet connection identified by the keyword ENET.

   ```
   CABINET 0
   PC AU02
   LED AM00
   BP D AC02
   S 0 GPNODE AK00 16 MRC 04
   S 1 GPNODE AK00 16 MRC 04 MIO H04 RAID3
   S 2 GPNODE AK00 16 MRC 04 MIO H04 RAID3
   S 3 GPNODE AN00 32 MRC 04 ENETMIO H04 RAID3 DAT
   S 4 GPNODE AK00 16 MRC 04
   .
   .
   .
   ```

   If *SYSCONFIG.TXT* is not correct, then you must edit *DEVCONF.TXT*.

2. Add lines for the additional I/O nodes and RAID subsystems. The additional lines are shown in **bold**.

```
DEVICES
ENET 00D03
RAID 00D03 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3
RAID 00D02 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 NOPAGER
RAID 00D01 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 NOPAGER
TAPE 00D03 ID 6 DAT
MIO 00D03 H04
MIO 00D02 H04
MIO 00D01 H04
END_DEVICES
```

Whether or not these new I/O nodes are to be used in the paging tree, identify them as NOPAGER in *DEVCONF.TXT*. This is because until their devices are built with **MAKEDEV** and until their disklabels are written, you must ensure that they are not built into a paging tree.

3. Connect to the Paragon system with the **console** command.

```
DS# cd /usr/paragon/boot
DS# ./console
        .
        .
        .
login:
```

4. Login as root to the Paragon system, unmount all file systems, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins.Then, issue **reset** with the **autocfg** option. This ensures that the new information in *DEVCONF.TXT* is added to *SYSCONFIG.TXT*.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS# ./reset autocfg
DS#
```

5. Check the */usr/paragon/boot/bootmagic* file to make sure the newly configured I/O nodes have been added to the BOOT_DISK_NODE_LIST line.

```
BOOT_DISK_NODE_LIST=7,15,23
```

6. Reset the Paragon system and enter multiuser mode. When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1.

```
DS# ./reset
        .
        .
        .
# <Ctrl-D>
```

7. Check the file */etc/devtab* on the Paragon system. This file must specify the boot node in two different ways: as an OS number and as a CBS (Cabinet-Backplane-Slot) number. The separators on the */dev/io0* line are tabs.

   Verify that */etc/devtab* does not have lines for any devices other than */dev/io0* (such as */dev/io1*, */dev/io2*). If it does, delete the extra lines and set IONODES = 1. The following example shows the correct format:

```
IONODES = 1
/dev/io0	7	0D3
```

8. Change to the */dev* directory. If */dev* contains any *io** directories other than *io0* (such as *io1*, *io2*), remove them. Do not delete */dev/io0*.

```
# cd /dev
```

9. Now, run **MAKEDEV**:

```
# ./MAKEDEV
```

   The **MAKEDEV** script creates the device special files */dev/io1*, */dev/io2*, and so on. Devices for the boot node are under */dev/io0* which is made automatically during installation of the system software. Then **MAKEDEV** uses the BOOT_DISK_NODE_LIST to check on the RAID level for each I/O node in the list. If it finds any RAID5 drives, it displays the following messages:

```
The RAID drives are either configured as RAID5, or are
otherwise in need of initialization. This procedure will
destroy any data currently on the drive. Do you want to
continue? (y/n)[n]
ARE YOU SURE? (y/n)[n]
```

   If you enter 'Y' or 'y' to both questions, the drive is initialized and converted to RAID3. If you do not convert a RAID5 drive to RAID3, the Paragon will not be able to access it. Note that it takes about 20 minutes to format a RAID subsystem.

10. Use the **disklabel** command to label each of the drives installed by **MAKEDEV**. For example, the following line labels the disk using the default boot-node, RAID3 disk label information from the */etc/disktab* file:

    ```
    # /usr/sbin/disklabel -rw /dev/io?/rz0a raid3
    ```

    where *?* is the number of the I/O node (1, 2, 3, ...) in the */dev/io?* directories. *raid3* is the disklabel.

    However, this boot node disklabel specifies that all file systems be made with only 8K-byte file system blocks. This is acceptable for UNIX OS binaries and program development, but is less than ideal if you intend to run typical supercomputer applications that perform large I/O operations with PFS.

    If the RAID subsystem you are labeling is on a non-boot node, choose another default label. Look in the */etc/disktab* file for provided labels. Some choices are as follows:

4gbraid3pfs	non-boot 4G-byte RAID used for PFS
4gbraid3pfspg	non-boot 4G-byte RAID used for PFS and paging
raid3pfs	non-boot 4.7G-byte RAID used for PFS
raid3pfspg	non-boot 4.7G-byte RAID used for PFS and paging

    If none of the provided disklabels fit your needs, you must customize your disklabel as follows:

    A. Use the **disklabel** command to read the default information into a file. The following command reads the disklabel from */dev/io1/rrz0a* which in this example turns out to be raid3pfspg.

    ```
    # /usr/sbin/disklabel -r /dev/io1/rrz0a > disklabelfile
    ```

    B. The following listing shows the label for raid3pfspg. This example shows four partitions: *a*, *b*, *c*, and *d*. *a* is a 20M-byte partition used for the PFS mount point. The *a* partition is used for the PFS mount point. The *b* partition is used for paging because it has an 8K-byte block size. The *c* and *d* partitions have 64K-byte block sizes and hence are used for PFS striping.

```
# /dev/io1/rrz0a:
type: SCSI
disk: raid3pfspg
label:
flags:
bytes/sector: 2048
sectors/track: 65
tracks/cylinder: 15
sectors/cylinder: 975
cylinders: 2480
```

```
rpm: 6300
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0           # milliseconds
track-to-track seek: 0  # milliseconds
drivedata: 0

4 partitions:
#        size   offset    fstype   [fsize bsize   cpg]
  a:    10240        0    4.2BSD     2048  8192    16    # (Cyl.    0 - 10*)
  b:  1048576    10240    4.2BSD     2048  8192    16    # (Cyl.   10*- 1085*)
  c:  1048576  1058816    4.2BSD     8192 65536    32    # (Cyl. 1085*- 2161*)
  d:   309248  2107392    4.2BSD     8192 65536    32    # (Cyl. 2161*- 2478*)
```

C. Use **disklabel** again to specify the new disk label based on the information from the file.

/usr/bin/disklabel -rR /dev/io1/rrz0a disklabelfile

11. After the drive is labeled, use **newfs** to create the file systems on the drive:

newfs -c 32 /dev/io?/rz0?

In this example, replace the *?* in */dev/io?* with the number of the I/O device (*1*, *2*, *3*, etc.), and replace the *?* in *rrz0?* with the partition name (in the example above, either *a*, *b*, *c*, or *d*). This means that the **newfs** command must be repeated for each partition on each device. The following script creates the file systems for partitions *a*, *b*, and *c* on a specified drive. If you try to newfs a non-existent partition, you get an error message.

```
#!/bin/sh
if [ $# -lt 1 ]
then
      echo "Usage: $0 <I/O node specification, e.g. io1>"
      exit 1
fi

partitions="a b c"

for part in $partitions
do
      echo "newfs -c 32 /dev/$1/rz0${part}"
      newfs -c 32 /dev/$1/rz0${part}
      echo ""
done
```

12. If the new I/O nodes are to be used in a paging tree, edit *DEVCONF.TXT* and remove the NOPAGER tokens. Then, issue a **reset autocfg** followed by a **reset**.

13. The partitions are now ready for mounting. To mount a partition create a mount point in the root partition, edit */etc/fstab* to include the mounting information, and reboot.

    For example, to mount a UFS file system on rz0c of the second I/O node with a mount point at */mio1*, create the directory in the root partition and add the following line to */etc/fstab*.

    ```
    /dev/io1/rz0c /mio1 ufs rw 0 4
    ```

    Be sure to set the access permission on the mount point that are appropriate for your setup.

Installing and Configuring the PFS

This section provides instructions for configuring a Parallel File System (PFS). The Paragon OSF/1 installation process configures and mounts a default PFS file system as specified in the default versions of the */etc/fstab* and */etc/pfstab* files.

The Default */etc/fstab* File

The */etc/fstab* file contains entries for file systems and disk partitions to be mounted at boot time. The */etc/pfstab* file contains definitions of the stripe groups. A stripe group consists of stripe directories where the actual PFS files reside.

The default configuration may not be the best configuration for your system. The default configuration is provided as a template that you can modify for your particular configuration.

NOTE

Running with the PFS striped across the boot node is not recommended, unless, of course, you have only one I/O node and it is the boot node.

The default configuration assumes that you have one I/O node. It mounts */*, */usr*, and */home* as UFS partitions. It mounts */pfs* as the PFS partition with stripe group *one*. The mounts are shown in */etc/fstab*, and the stripe groups are defined in */etc/pfstab*.

When looking at */etc/fstab*, note that *io0* has partitions *rz0a*, *rz0b*, *rz0c*, *rz0d*, *rz0e*, *rz0f*, and *rz0g*.

- The partitions *rz0b*, *rz0c*, and *rz0g* are not mounted. They are not mentioned in */etc/fstab*.

- The partitions *rz0a*, *rz0e*, and *rz0f* are UFS file systems. *rz0a* is the root partition */*; *rz0e* is the user partition */usr*; and *rz0f* is the home partition */home*. This information is embodied in the following lines from */etc/fstab*.

```
/dev/io0/rz0a   /                  ufs rw 0 1
/dev/io0/rz0e   /usr               ufs rw 0 2
/dev/io0/rz0f   /home              ufs rw 0 3
```

- The partition *rz0d* is the mount point for the PFS file system, and its stripe group is one. This means that */etc/pfstab* must have the stripe group *one* defined. This information is embodied in the following line from */etc/fstab*.

```
/dev/io0/rz0d   /pfs               pfs rw,stripegroup=one 0 3
```

Here is the complete text of the default */etc/fstab*. The # at the start of a line indicates a commented line.

```
# Local (required to boot mesh) filesystems
#
/dev/io0/rz0a    /                         ufs rw 0 1
/dev/io0/rz0e    /usr                      ufs rw 0 2
#
# Additional local filesystems
#
/dev/io0/rz0f    /home                     ufs rw 0 3
#
# Remote filesystems
#/dev/io1/rz0c   /home/.sdirs/vol0         ufs rw 0 5
#/dev/io2/rz0c   /home/.sdirs/vol1         ufs rw 0 5
#/dev/io3/rz0c   /home/.sdirs/vol2         ufs rw 0 5
#/dev/io4/rz0c   /home/.sdirs/vol3         ufs rw 0 5
#
# Parallel filesystems
#
/dev/io0/rz0d   /pfs                    pfs rw,stripegroup=one 0 3
#
# NFS filesystems
#
```

The Default */etc/pfstab* File

When looking at the default */etc/pfstab*, notice there are nine stripe groups defined. The stripe groups *eight* and *all* are the same. Here is the default */etc/pfstab*. Note that stripe group *one* consists of the directory */home/.sdirs/vol0*. */home* is mounted on partition *rz0f*. Please note that the names *one* through *eight* are only examples. You can use any group names you like.

```
/home/.sdirs/vol0 all one two three four five six seven eight
/home/.sdirs/vol1 all     two three four five six seven eight
/home/.sdirs/vol2 all         three four five six seven eight
/home/.sdirs/vol3 all               four five six seven eight
/home/.sdirs/vol4 all                    five six seven eight
/home/.sdirs/vol5 all                         six seven eight
/home/.sdirs/vol6 all                             seven eight
/home/.sdirs/vol7 all                                   eight
```

Even with one I/O node, it may make sense to have two stripe directories. A stripe directory is in a UFS file system and cannot contain files larger than 2G-1 bytes. For PFS files to be larger than 2G-1 bytes, they must be striped across more than one stripe directory, and each stripe directory must be a separate file system. Then, the portion of a PFS file in a stripe directory is always less than 2G-1 bytes while the entire file may be larger.

Additional I/O Nodes

When you configure the PFS, you may choose to have different I/O nodes used by the PFS and the paging tree. To specify that an I/O node is *not* to be used by the paging tree, add the NOPAGER token to the RAID line in *DEVCONF.TXT*. Then, perform a **reset autocfg**. For example, the following line shows how to identify the I/O with CBS number 00D01 as an I/O node that will not be used for paging.

```
RAID 00D01 ID 0 SW 3.02 LV 3 DC 5 SID 0 RAID3 NOPAGER
```

To configure additional I/O nodes for PFS,

1. Follow the instructions in the previous section under "Configuring Additional RAID Subsystems" on page 2-38 to configure file systems on I/O nodes other than the boot node.

   The partitioning of the RAID subsystem is determined by the disk label. Choose either the non-boot *pfs label or the non-boot *pfspg label.

2. Uncomment or add the appropriate entries in */etc/fstab*. For example, if four I/O nodes are added for PFS file striping, the */etc/fstab* entries might be changed to the following:

```
# Local (required to boot mesh) filesystems
#
/dev/io0/rz0a    /                         ufs rw 0 1
/dev/io0/rz0e    /usr                      ufs rw 0 2
#
# Additional local filesystems
#
/dev/io0/rz0f    /home                     ufs rw 0 3
#
# Remote filesystems
/dev/io1/rz0c   /home/.sdirs/vol0         ufs rw 0 5
/dev/io2/rz0c   /home/.sdirs/vol1         ufs rw 0 5
/dev/io3/rz0c   /home/.sdirs/vol2         ufs rw 0 5
/dev/io4/rz0c   /home/.sdirs/vol3         ufs rw 0 5
#
# Parallel filesystems
#
/dev/io0/rz0d   /pfs                pfs rw,stripegroup=four 0 3
#
# NFS filesystems
```

   The difference from the default */etc/fstab* is that the remote file systems are uncommented and the stripe group has been changed to four.

The new configuration has the following consequences:

- The stripe group of the PFS mount has been changed from *one* to *four*. Files created in */pfs* will be striped across stripe group *four*, which is defined in */etc/pfstab* to consist of the directories */home/.sdirs/vol0*, */home/.sdirs/vol1*, */home/.sdirs/vol2*, and */home/.sdirs/vol3*. Because each of the stripe directories is the mount point of a UFS file system on a different I/O node, concurrent striping to four I/O nodes is achieved.

- PFS files are not striped across the boot node, *io0*.

- 2G-1 bytes of each RAID subsystem are used. This is because the stripe directory is mounted on a UFS file system, which has a limit of 2G-1 bytes. This means that maximum use of the data storage capability of the RAID subsystems for PFS use is not attained. However, concurrency is maximized.

  If you decide to have two stripe directories per RAID subsystem, the remote mounts in */etc/fstab* might look as follows:

```
#
# Remote filesystems
#
/dev/io1/rz0c /home/.sdirs/vol0 ufs rw 0 5
/dev/io2/rz0c /home/.sdirs/vol1 ufs rw 0 5
/dev/io3/rz0c /home/.sdirs/vol2 ufs rw 0 5
/dev/io4/rz0c /home/.sdirs/vol3 ufs rw 0 5
#
/dev/io1/rz0d /home/.sdirs/vol4 ufs rw 0 5
/dev/io2/rz0d /home/.sdirs/vol5 ufs rw 0 5
/dev/io3/rz0d /home/.sdirs/vol6 ufs rw 0 5
/dev/io4/rz0d /home/.sdirs/vol7 ufs rw 0 5
#
# Parallel filesystems
#
/dev/io0/rz0d /pfs pfs rw,stripegroup=eight 0 3
```

With this configuration, a 128K-byte write will be split between *io1* and *io2*.

The stripe directories must be owned by *root* and have write/execute permissions for *root*, write/execute permissions for *group* and *other*, and have their sticky bits set. The PFS mount point must have read/write/execute permissions for everyone and have its sticky bit set.

The file systems must be mounted before you can change their ownership or permissions. To mount the file systems listed in */etc/fstab*, issue the following command:

\# ***mount -a***

The commands to set the ownership and permissions are as follows (the leading 1 sets the sticky bit):

\# ***chown root*** *stripe_directoriues*
\# ***chmod 1333*** *stripe_directoriues*

The command to do this is as follows:

```
# chmod 1777 mount_point
```

Setting up a Paging Tree

Without a paging tree, all of the compute nodes in your system page to the boot node. This is the default and is called a two-level paging tree. Using the default for large systems may result in poor performance.

If your Paragon system has I/O nodes in addition to the boot node, you can use a three-level paging tree to designate these additional I/O nodes as paging nodes.

The following diagram shows a three-level paging tree in which two I/O paging nodes (15 and 23) accept paging from the compute nodes. Nodes 15 and 23 in turn page to the boot node (7). It is called three-level because the first level consists of the compute nodes paging to the MIO nodes; the second level is the MIO nodes paging to the boot node; the third level is the boot node paging to the RAID array.

```
 0...63   Compute Nodes               64...127  Compute Nodes
(except 7,15,23)
    \         /                                 \       /
     \       /                                   \     /
       15   MIO Node                            23   MIO Node
        |                                                |
        |  \                                           / |
        |   \                                         /  |
       RAID    +------------+--------------+            RAID
       Array                |                           Array
                            |
                            7 Boot Node
                            |
                            |
                            RAID Array
```

A three-level paging tree (as shown in the above diagram) where all the non-boot I/O nodes page to the boot node can be obtained by specifying the **-P1** option for **bootpp** in the **reset** script in */usr/paragon/boot* on the diagnostic station. The default paging partition is *rz0b*.

1. Log into the diagnostic station as *root*. Ensure that the partitions *rz0b* on your RAID subsystems do not have file systems mounted on them. Note that, unless you change the default paging partition, all data on *rz0b* will be lost.

2. Ensure that the device nodes for all the RAID subsystems are created. For example, if you have three MIO nodes, the following device nodes must exist on the Paragon system in */dev*: *io0*, *io1*, and *io2*.

3. The **reset** script in */usr/paragon/boot* contains the following line:

```
$BOOTPP -P 1 -W -Z\
```

This method defines the default paging partition on the I/O nodes as *rz0b*. The paging tree is created when reset your system. To not have a paging tree remove the option **-P 1**.

Do not issue **bootpp** from the command line. The proper procedure is to edit the **reset** script and reset your system.

To have an I/O node use a paging partition other than the default, perform the following steps.

A. Edit *DEVCONF.TXT* and put the PAGE_TO token on the RAID line identifying the I/O node that will do the paging. If PAGE_TO is left out the default *rz0b* is used. For example, the following line shows how to identify the I/O with CBS number 00A01 as a paging node that uses the *rz0c* partition for paging.

```
RAID 00A01 ID 0 SW 3.04 LV 3 DC 5 SID 0  RAID3 PAGE_TO rz0c
```

B. Then issue a **reset autocfg**. This creates a new *SYSCONFIG.TXT*. It's best to put the PAGE_TO token in *DEVCONF.TXT* because if it's not there, then whenever you do a **reset autocfg**, you will lose the paging information.

C. Another way (which is not the recommended way) is to set the bootmagic variable PAGE_TO in */usr/paragon/boot/MAGIC.MASTER*. If you then perform a **reset**, the specification in *MAGIC.MASTER* overrides that in *DEVCONF.TXT* or *SYSCONFIG.TXT*. The format of this variable is

```
PAGE_TO=<node_nbr, node_nbr, . . . >part:<node_nbr, node_nbr, . . . >part. .
```

For example, if you wanted the paging tree shown in the diagram for this example to page to partition *rz0c* instead of *rz0b*, add the following line to *usr/paragon/boot/MAGIC.MASTER*:

```
PAGE_TO=<15>rz0c:<23>rz0c
```

NOTE

In the *SYSCONFIG.TXT*, *MAGIC.MASTER*, or *DEVCONF.TXT* files, the PAGE_TO specification has no effect on the boot node. The boot node always uses the default *rz0b* as the paging partition.

4. Connect to the Paragon system with the **console** command.

```
DS# cd /usr/paragon/boot
DS# ./console
        .
        .
        .
login:
```

5. Login as *root* to the Paragon system, unmount all file systems, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins. Then, use the **reset** script to reboot the system.

   When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if in the *MAGIC.MASTER* file contains RB_MULTIUSER=1.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS# ./reset
        .
        .
        .
# <Ctrl-D>
```

   bootpp performs the following: (Note that in what follows the node numbers obey the OS numbering scheme, and not the CBS numbering scheme.)

   - If the */usr/paragon/boot/MAGIC.MASTER* file does not define BOOT_DISK_NODE_LIST, **bootpp** adds a definition to */usr/paragon/boot/bootmagic* based on information in *SYSCONFIG.TXT*.

     In this example, the BOOT_DISK_NODE_LIST line looks like:

     ```
     BOOT_DISK_NODE_LIST=7,15,23
     ```

     Note that if BOOT_DISK_NODE_LIST is defined in *MAGIC.MASTER*, **bootpp** uses that value when constructing *bootmagic*.

   - If the */usr/paragon/boot/MAGIC.MASTER* file does not define EXPORT_PAGING and PAGER_NODE, **bootpp** adds definitions based on information in *SYSCONFIG.TXT*. In this example, the definitions are as follows:

```
EXPORT_PAGING=7,15,23
PAGER_NODE=<0..6>15:<8..14>15:<15>7:<16..22>15:<23>7:<24..63>15:<64..127>23
```

Note that all the nodes in the EXPORT_PAGING list (except the boot node) are importing their paging from the boot node. The rest of the nodes import their paging from the new paging nodes that are importing their paging from the boot node.

Also note that if EXPORT_PAGING and PAGER_NODE are defined in *MAGIC.MASTER*, **bootpp** uses those values when constructing *bootmagic*.

Increasing Default Paging File Size

By default, the **install** script creates a 64M-byte page file size for the default pager on the boot node. If you want to increase the size of the paging file after installation, follow the instructions in this section.

If the system produces an error similar to `Paging File Exhausted`, it could indicate the paging size is too small. The following procedure shows how to increase the page size which, in this example, is increased to 512M bytes. The new paging file is placed in the */home* partition because the */root* partition is not large enough for such a file.

NOTE

Increasing the page size to 512M bytes can take up to thirty minutes. A smaller page size will take less time. Before you increase page size, you should make sure there is enough room in the home partition.

1. Log in or **rlogin** to the diagnostic station as *root.*

2. Connect to the Paragon system with the **console** command. The **console** command is a script that the **reset** script created in */usr/paragon/boot* the last time **reset** was run. **console** uses whatever the console connection was at the time and supports the **async**, **fscan**, and the **scanio** console interfaces. You may have to press a carriage return after issuing **console** to get a prompt.

```
DS# cd /usr/paragon/boot
DS# ./console
        .
        .
login:
```

3. Login as *root* to the Paragon system, unmount all file systems, and use the **halt** command to shut down the Paragon system. Then, return to the diagnostic station prompt by typing **~.** (or **~~.** if you are logged in remotely to the diagnostic station). The key sequence **~q** also works and does not require that you keep track of the number of remote logins. Then, use the **reset** script to reboot the system.

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
# ~.
DS#
```

When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1.

4. Now, issue a **reset ramdisk** and perform the following commands.

```
DS# ./reset ramdisk
<ramdisk> mount -u /
<ramdisk> fsck -y /dev/rrz0a
<ramdisk> mount -w /dev/rz0a /root
<ramdisk> fsck -y /dev/rrz0f
<ramdisk> mount -w /dev/rz0f /home
<ramdisk> cd /home
<ramdisk> /root/sbin/create_pf 512M paging_file
<ramdisk> chmod 600 paging_file
<ramdisk> cd /root/dev
<ramdisk> ln io0/rz0f rz0f
<ramdisk> cd /root/mach_servers
<ramdisk> mv paging_file paging_file.orig
<ramdisk> /root/sbin/ln -s /dev/rz0f/paging_file paging_file
<ramdisk> cd /
<ramdisk> sync
<ramdisk> umount /root
<ramdisk> umount /home
<ramdisk> ~.
DS# ./reset
```

When the **reset** command has completed and the prompt returns, enter multiuser mode by pressing **<Ctrl-D>**. Note that the **reset** script automatically enters multiuser mode if the *MAGIC.MASTER* file contains RB_MULTIUSER=1.

Installing the Paragon™ System Acceptance Tests

These instructions assume that you have copied the compressed **tar** files from the distribution tape to the diagnostic station or to some other server.

1. Log in to the Paragon system as *root.*

2. Establish an **ftp** connection with the server containing the files, *sat.tar.Z* and *sat.doc.tar.Z.*

```
# cd /tmp
# ftp IP address of server with distribution files
Name: login_name
Password:password
ftp> cd path to distribution files
ftp> bin
ftp> get sat.tar.Z
ftp> get sat.doc.tar.Z
ftp> bye
```

3. Use the following steps to uncompress and untar the distribution files into the / directory.

```
# uncompress sat.tar.Z
# cd /
# tar xf /tmp/sat.tar
# rm /tmp/sat.tar
# cd /usr/share
# tar xf /tmp/sat.doc.tar
# rm /tmp/sat.doc.tar
```

If you are having disk space problems, use **zcat** to uncompress and extract. This uses less space, but takes longer:

```
# cd /
# zcat /tmp/sat.tar.Z | tar xvf -
# rm /tmp/sat.tar.Z
# cd /usr/share
# tar xf /tmp/sat.doc.tar
# rm /tmp/sat.doc.tar
```

Before running the SATs, you must configure a Parallel File System (PFS). Refer to "Installing and Configuring the PFS" on page 2-44 for instructions on configuring a PFS.

The Network Queuing System (NQS)

If you will be running NQS at your site, refer to Appendix A of the *Paragon™ Network Queueing System Manual* for configuration instructions.

The Multi-User Accounting and Control System (MACS)

If you will be running MACS at your site, refer to Chapter 4 of the *Paragon™ Network Queueing System Manual* for configuration instructions.

Understanding Node Numbering

There are two node numbering schemes on the Paragon system: *CBS* (Cabinet:Backplane:Slot) numbering, and *OS* numbering (also called *root partition* numbering).

You should become familiar with both schemes, because you will encounter them both. For example, the *SYSCONFIG.TXT* file uses CBS numbering because it is primarily a hardware configuration file, and the *MAGIC.MASTER* file uses OS numbering because it is primarily a software configuration file.

CBS Numbering

To identify a node with CBS numbering, you specify its cabinet, backplane, and slot.The following display shows the first few lines of an example *SYSCONFIG.TXT* file, which illustrates how the CBS numbers are used: the first entry indicates cabinet 0, the BP indicates backplane D, and the multiple S's indicate Slots 0 through 15.

```
CABINET 0
PC AU02
LED AM00
BP A AC02
S  0 EMPTY
S  1 GPNODE AN00 32 MRC 04  MIO H04 RAID3
S  2 EMPTY
S  3 EMPTY
S  4 GPNODE AK00 16 MRC 04
S  5 GPNODE AK00 16 MRC 04
S  6 GPNODE AK00 16 MRC 04
S  7 GPNODE AK00 16 MRC 04
S  8 GPNODE AK00 16 MRC 04
S  9 GPNODE AK00 16 MRC 04
S 10 GPNODE AK00 16 MRC 04
S 11 GPNODE AK00 16 MRC 04
S 12 GPNODE AK00 16 MRC 04
```

```
S 13 GPNODE AK00 16 MRC 04
S 14 GPNODE AK00 16 MRC 04
S 15 GPNODE AK00 16 MRC 04
```

Cabinets

Cabinet numbering begins with Cabinet 0 which is the right-most cabinet as viewed from the front.

Backplanes

There are up to four backplanes in a cabinet. The backplanes are identified alphabetically, starting with A at the bottom. Each backplane consists of up to 16 nodes, which are arranged linearly in slots.

Slots

When the nodes are viewed from the front, the slot numbering starts at the right of the card cage and moves left. To find the LED associated with a slot, you begin counting at the lower-right corner of the backplane's block of LEDs, count up the right column, and then over to the next column to the left, and then back down in a serpentine fashion; that is, Slot 0 is the lower-right LED, and Slot 15 is the lower-left LED.

The following shows how the slots in a single backplane map to the LEDs in the cabinet door.

```
15   14   13   12   11   10   9   8 <perf>  7   6   5   4   3   2   1   0

Physical layout of slots in card cage (front view)
The <perf> indicates the empty performance monitor slot.



12      11      4     3
13      10      5     2
14       9      6     1
15       8      7     0

Mesh arrangement of slots as shown on LED panel (front view)
```

OS Numbering

OS numbering starts at the top left of the left-most cabinet, and spans all of the cabinets in the Paragon system. When you reach the end of the right-most cabinet, return to the left-most cabinet, drop down a row, and continue counting. OS node numbering is also called root partition node numbering.

The following shows the OS numbers and the CBS numbers for all the nodes in a two-cabinet system. A row of LEDs is represented by two rows of numbers. The top row of numbers contains the OS numbers and the bottom row contains the CBS numbers. For example the node with an OS number of 21 has a CBS number of 0D09.

```
    Cabinet 1                 |         Cabinet 0
0      1      2      3      | 4      5      6      7     OS  number
1D12   1D11   1D04   1D03   | 0D12   0D11   0D04   0D03  CBS number
8      9      10     11     | 12     13     14     15
1D13   1D10   1D05   1D02   | 0D13   0D10   0D05   0D02
16     17     18     19     | 20     21     22     23      Backplane D
1D14   1D09   1D06   1D01   | 0D14   0D09   0D06   0D01
24     25     26     27     | 28     29     30     31
1D15   1D08   1D07   1D00   | 0D15   0D08   0D07   0D00
==================================================
32     33     34     35     | 36     37     38     39
1C12   1C11   1C04   1C03   | 0C12   0C11   0C04   0C03
40     41     42     43     | 44     45     46     47
1C13   1C10   1C05   1C02   | 0C13   0C10   0C05   0C02
48     49     50     51     | 52     53     54     55      Backplane C
1C14   1C09   1C06   1C01   | 0C14   0C09   0C06   0C01
56     57     58     59     | 60     61     62     63
1C15   1C08   1C07   1C00   | 0C15   0C08   0C07   0C00
==================================================
64     65     66     67     | 68     69     70     71
1B12   1B11   1B04   1B03   | 0B12   0B11   0B04   0B03
72     73     74     75     | 76     77     78     79
1B13   1B10   1B05   1B02   | 0B13   0B10   0B05   0B02
80     81     82     83     | 84     85     86     87      Backplane B
1B14   1B09   1B06   1B01   | 0B14   0B09   0B06   0B01
88     89     90     91     | 92     93     94     95
1B15   1B08   1B07   1B00   | 0B15   0B08   0B07   0B00
==================================================
96     97     98     99     | 100    101    102    103
1A12   1A11   1A04   1A03   | 0A12   0A11   0A04   0A03
104    105    106    107    | 108    109    110    111
1A13   1A10   1A05   1A02   | 0A13   0A10   0A05   0A02
112    113    114    115    | 116    117    118    119     Backplane A
1A14   1A09   1A06   1A01   | 0A14   0A09   0A06   0A01
120    121    122    123    | 124    125    126    127
1A15   1A08   1A07   1A00   | 0A15   0A08   0A07   0A00
==================================================
```

Parallel File System Performance 3

Current PFS Performance

This section attempts to outline some of the PFS performance characteristics observed in Paragon OSF/1 R1.2, so that systems may be configured appropriately for their application set. All benchmarks were run using 32 MB I/O nodes with the message coprocessor enabled. Also, in order to dedicate more memory to the Mach IPC subsystem, the following *page list* boot magic configuration variables were added to the `MAGIC.MASTER` file on the Diagnostic Station:

```
NETIPC_PGLIST_HIGH=103
NETIPC_PGLIST_LOW=73
NETIPC_PGLIST_REFILL=85
```

These boot magic variables are described in the *Paragon™ System Software Release 1.2 Release Notes*.

NOTE

The data presented in this section are subject to fluctuation, since modifications made to other subsystems (e.g., the Mach IPC subsystem) can radically influence I/O throughput. This section will be updated periodically with the most recent "snapshots" of system performance.

Single Compute Node

Assuming a base file system block size of 64 KB for PFS stripe data storage, and a request size of at least 64 KB, PFS maintains a write throughput of about 2.6 MB/sec and a read throughput of about 2.9 MB/sec striping to a single I/O node. Throughput is a few hundred KB/sec higher than this if the application and file system are co-located on the same node. There is little room for improvement here with the current I/O hardware; direct disk access is not much faster.

Figure 3-1 illustrates PFS write bandwidth from an application running on one compute node, using request sizes ranging from 8 KB to 1 MB, with samples every 64 KB. The size of the file used was 64 MB. The file system block size was 64 KB, as was the stripe unit size. Eight I/O nodes were used, so the stripe factor was varied from one to eight. All tests were run on nodes with 32 MB of RAM. Each data point is an average of three separate runs. For comparison with PFS, the lowest line on the graph shows current UFS performance.

As this figure shows, PFS write bandwidth scales well up to five I/O nodes (vs. four in R1.1), to

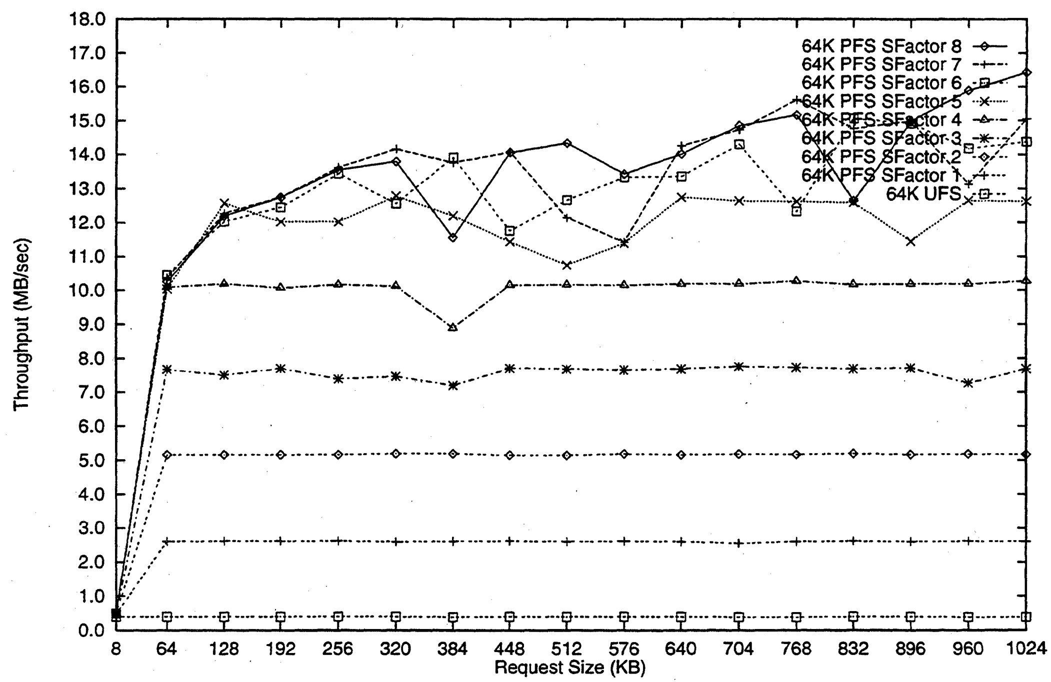

Figure 3-1. PFS Write Performance (1 Compute Node)

about 12 MB/sec (peak 12.8 MB/sec). Currently, adding more I/O nodes has a limited payoff when writing from only one compute node due to Mach IPC bandwidth limitations. Mach IPC must be used by PFS for OS-level message traffic. Also, when writing to more than five I/O nodes concurrently Mach IPC performance becomes unpredictable, and bandwidth varies more radically when the request size is changed due to timing issues in the IPC implementation. However, the bandwidth does not get worse as I/O nodes are added, and a peak bandwidth of 16.4 MB/sec was reached using eight I/O nodes with a request size of 1 MB.

Note that write performance quickly increases to high speeds at the 64 KB request size, even though the optimal request size is a multiple of one file stripe (256 KB for SFactor 4) so that all I/O nodes are written to concurrently. This speedup at 64 KB occurs because, due to the fact that writes to disk are performed asynchronously, overlapped disk I/O is allowed to occur to different I/O nodes across multiple write requests from the application. This speedup does not occur on reads: reads scale up fairly linearly until the 256 KB request size.

PFS read bandwidth is displayed in Figure 3-2. Here, the Mach IPC scaling problem is exacerbated by what is known as the "many-to-one" problem. In the read case, PFS on the compute node gathers incoming data from multiple I/O nodes concurrently, and unfortunately the protocol used by Mach IPC does not handle this "fan-in" case well. For example, often the IPC layer cannot buffer all the incoming requests, and the less-than-optimal flow control mechanisms force the senders to resend the data.

As a result, as more I/O nodes are added read performance is less predictable then write

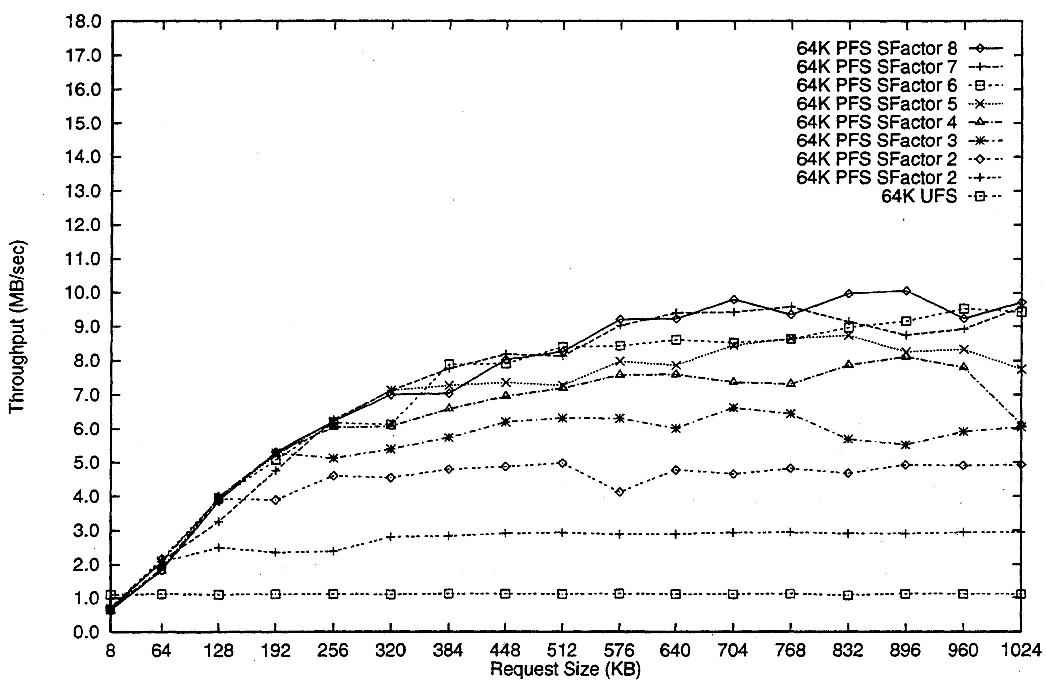

Figure 3-2. PFS Read Performance (1 Compute Node)

performance, and is very sensitive to request size due to Mach IPC timing issues. Read bandwidth does not scale as linearly as writes when I/O nodes are added: generally a bandwidth of 7-8 MB/sec can be achieved with four I/O nodes. Peak read performance observed was 10.1 MB/sec from 8 I/O nodes.

Note in these graphs that the performance obtained in a regular UFS file system does not improve as the request size is increased. This is due to the fact that UFS maps the file into compute node memory for caching purposes, and uses the Virtual Memory system to page file data in and out rather than explicitly calling on the file system to read and write file data. The current VM system in the Mach microkernel only handles one page at a time, rather than using *page clusters* where appropriate, so the actual disk I/O is limited to one 8 KB transfer at a time no matter what the user request size is. In contrast, although PFS stores file data in UFS file systems, it bypasses the UFS caching code and performs I/O directly to/from disk.

For comparison with the recommended 64 KB file system block size, Figure 3-3 and Figure 3-4 illustrate write and read performance, respectively, when a block size of 8 KB is used on all file

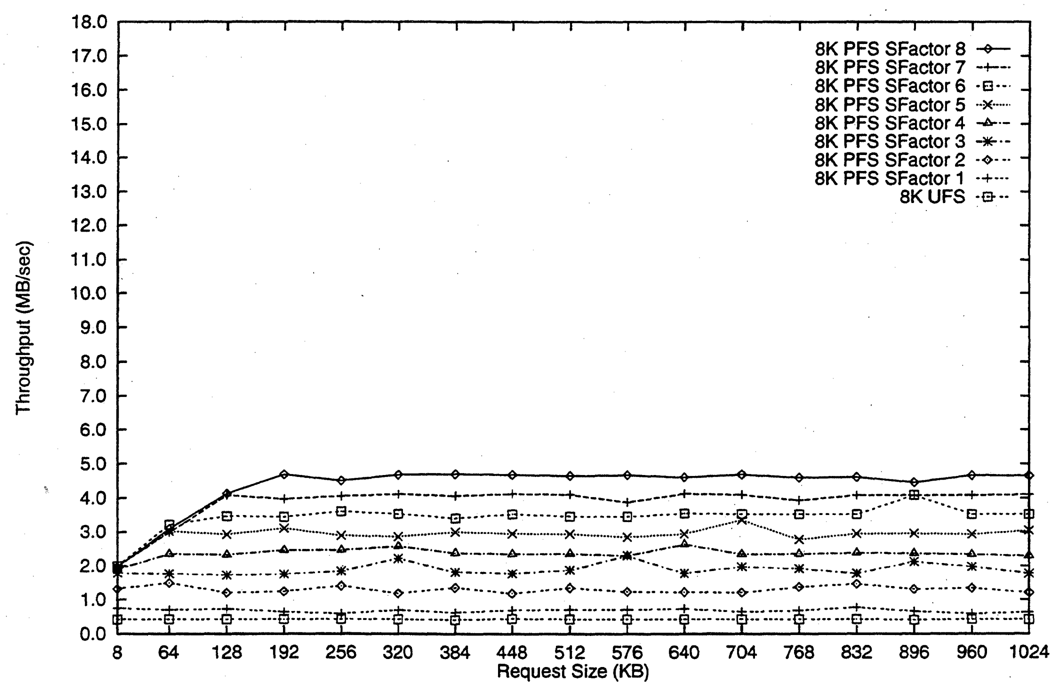

Figure 3-3. 8 KB PFS Write Performance (1 Compute Node)

systems. The PFS stripe unit size is also 8 KB in these tests. In this case, reads generally outperform writes because disk block coalescing is performed in the read case, so that fewer I/O's to disk are performed, and this has a greater effect at the smaller block size. Again, reads currently don't scale past four I/O nodes, although writes do since we are not pushing against Mach IPC bandwidth limitations at this block size (the speed of the file system-to-disk I/O is the limiting factor here), and since writes from one compute node do not suffer from the Mach IPC "many-to-one" problem.

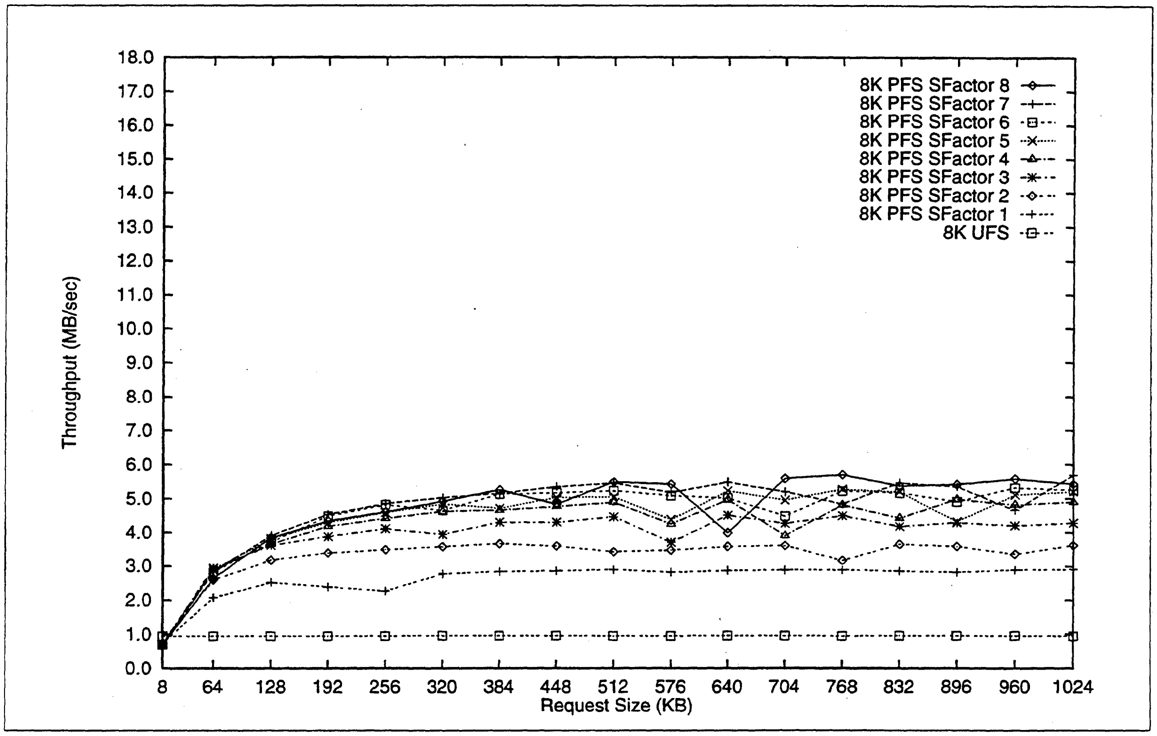

Figure 3-4. 8 KB PFS Read Performance (1 Compute Node)

Multiple Compute Nodes

Regrettably, a large system was not available for the extended length of time required to perform a complete performance analysis using multiple compute nodes. Detailed results are presented here for a system with a 1:1 ratio of 8 compute nodes and 8 I/O nodes. This section will be updated with results from larger systems at larger ratios when resources become available.

Figure 3-5 shows write performance results to a 64 KB PFS file system striped across 8 I/O nodes.

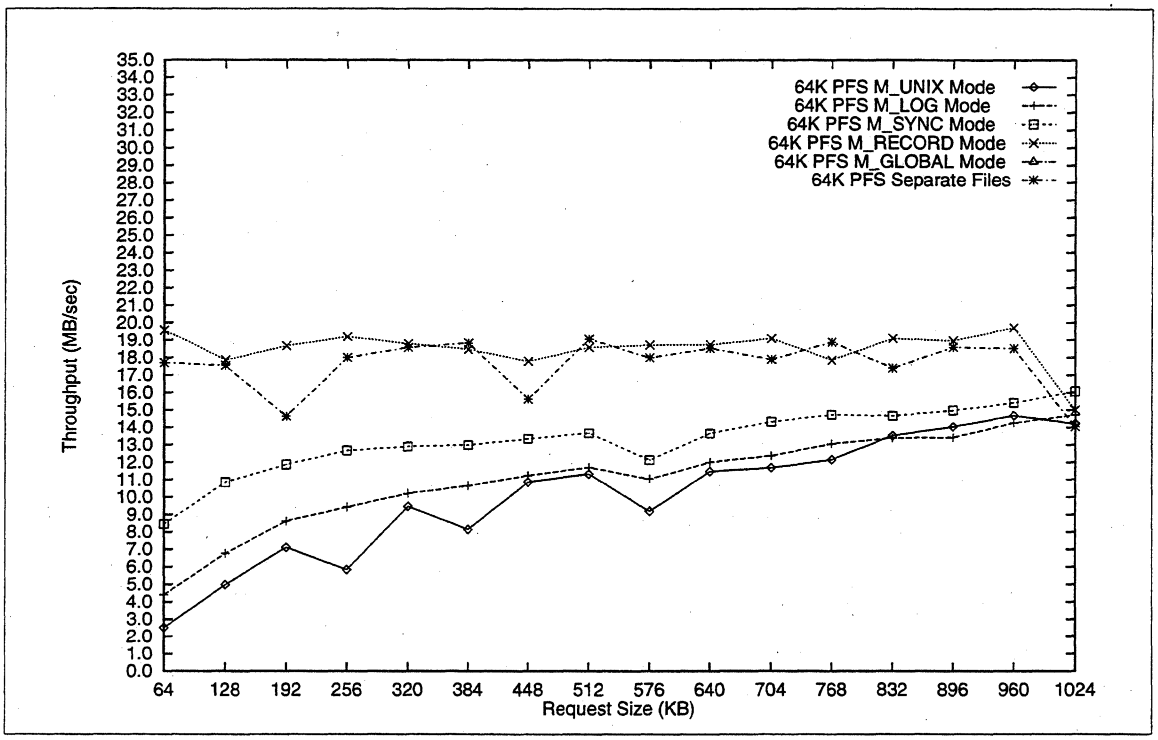

Figure 3-5. 64 KB PFS Write Performance (8 Compute Nodes, 8 I/O Nodes)

Each line in the graph represents a different I/O mode. For comparison, performance is also plotted for the "separate files" case, where each compute node accesses a separate file in the same PFS file system rather than all nodes sharing the same file.

The **M_UNIX**, **M_LOG**, and **M_SYNC** modes all have similar performance. These modes all allow only single-writers to the file at a time, so this in effect sequentializes all writes to the file because only one compute node is allowed to access the file at any one time. Thus, performance here should be similar to what was obtained on the single-compute node tests, which is the case. Note that the **M_SYNC** mode performs somewhat better than the other two modes: this is due to optimized file token management in this mode.[1]

The **M_RECORD** mode allows writes to occur in parallel since it enforces the rule that each write is targeted for a distinct file record. Thus **M_RECORD** is able to maintain a fairly consistent rate of 18-19 MB/sec on 8 I/O nodes. The "Separate Files" achieves similar performance because the single-writer rule does not need to be enforced when each node is writing to a separate file. Using separate files does, of course, impose more file open and close overhead, and moves more file management burden to the application.

1. Further discussion of file tokens is beyond the scope of this document.

The **M_GLOBAL** mode does not appear on this graph because bandwidth numbers for **M_GLOBAL** writes are fairly useless (the line is actually way above the others, off the graph, in the 100 MB/sec range). We don't bother to show these numbers because by definition, in **M_GLOBAL** mode all nodes write the same data, so the operating system simply throws out all data except that from one node. The true aggregate bandwidth calculated is then the bandwidth achieved by this one node multiplied by the number of nodes, which is quite high, even though the true "disk bandwidth" achieved (by the one node) is in the 18-20 MB/sec range.

The I/O mode read data is displayed in Figure 3-6. Again, the **M_UNIX**, **M_LOG**, and **M_SYNC**

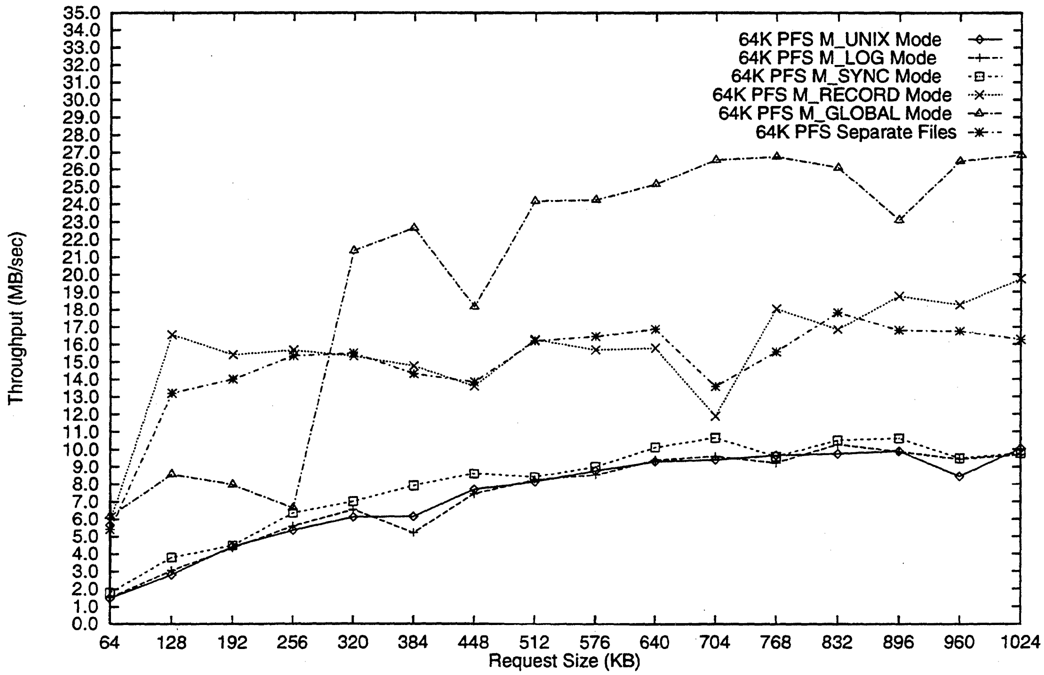

Figure 3-6. 64 KB PFS Read Performance (8 Compute Nodes, 8 I/O Nodes

bandwidths are very similar. However, here it should be especially noted that even in the read case, I/O is sequentialized in these modes. In other words, the Paragon file system *does not support multiple readers*, even in **M_UNIX** mode. This is a fairly serious deficiency in the implementation which hopefully will be fixed in a future release. Multiple readers/single writer should be supported. Until this is supported, performance in these modes should be similar to what was obtained on the single-compute node tests, which is the case.

M_RECORD does support multiple readers, which explains the higher performance of this mode. Read performance in this mode actually improves when the compute:I/O ratio is increased to 8:1, however there was no system available to provide detailed analysis for this case. Similar to writes, the "Separate Files" case achieves read performance similar to that of **M_RECORD**.

The **M_GLOBAL** performance data is more meaningful for reads, and as the graph shows in this configuration that it is able to achieve 27 MB/sec at some request sizes. The major limitation on performance in this mode is Mach IPC bandwidth; we expect these numbers to greatly improve when the NORMA rewrite is integrated into the operating system. Performance is somewhat erratic in this mode currently, probably due to the effects of paging in the same read data on multiple compute nodes, however little time has been invested at this point in analyzing what's going on. Note again that aggregate **M_GLOBAL** bandwidth numbers do not necessarily reflect "true" disk performance, rather they reflect how fast we can pump the *same* file data from disk to *all* compute nodes using Mach IPC.

Figure 3-7 and Figure 3-8 display the same data as the previous two figures, except that data points

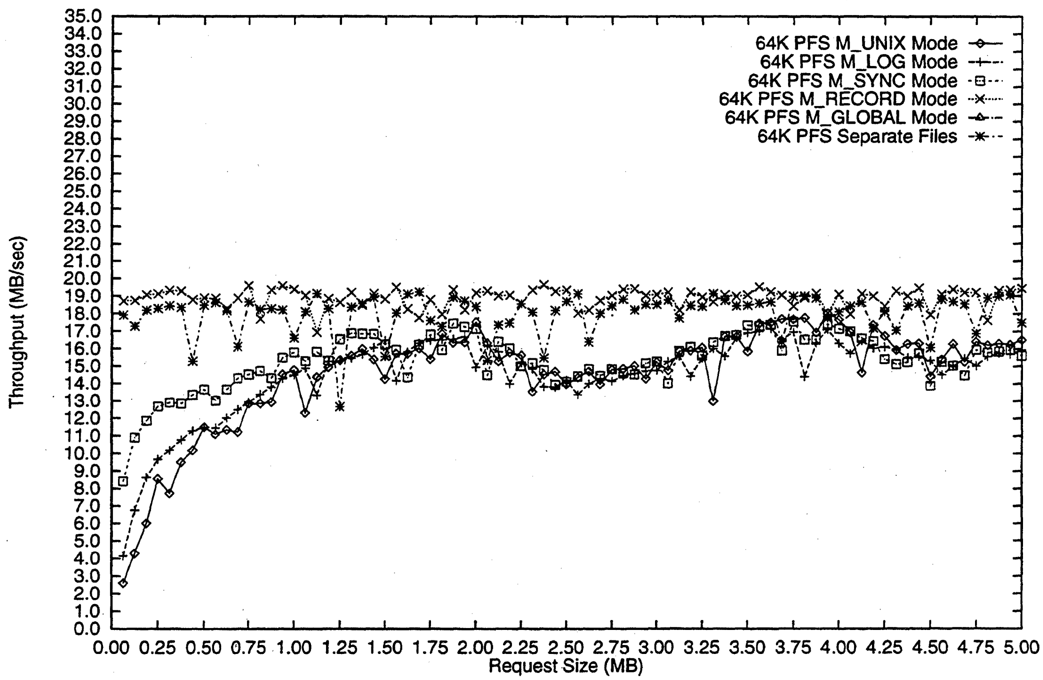

Figure 3-7. 64 KB PFS Write Performance (8 Compute Nodes, 8 I/O Nodes)

for I/O request sizes up to 5 MB rather than 1 MB are shown. Note the anomaly reading in the **M_GLOBAL** mode at a request size of 4 MB or greater in this configuration; this is undoubtedly due to a system paging problem since the bandwidth rate plummets to exactly the system page-in rate from a single pager node. However, the exact cause is not yet known.

On larger systems, and at larger compute:I/O node ratios, the **M_RECORD**, **M_GLOBAL**, and

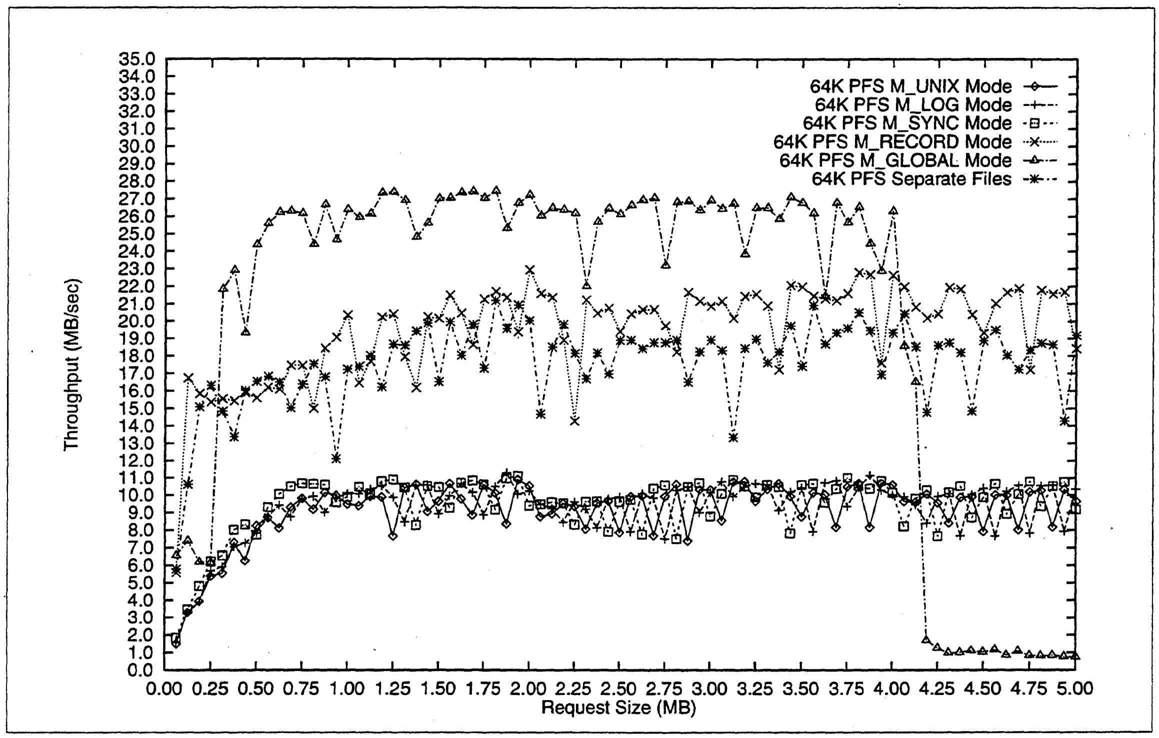

Figure 3-8. 64 KB PFS Read Performance (8 Compute Nodes, 8 I/O Nodes)

"Separate Files" bandwidth graphs should scale up fairly linearly with the number of I/O nodes. This has indeed been verified to some degree, although not enough analysis on larger systems has been done at this point in time to warrant a detailed graph such as those presented here. Peak bandwidth reported at this point in time is 94 MB/sec writing to one PFS file striped across 30 I/O nodes, using the **M_RECORD** mode.

On larger systems, the **M_UNIX**, **M_LOG**, and **M_SYNC** bandwidth graphs should see limited improvement, since these modes only support single reader/single writer access to the file. Improvement in this modes will not occur until the NORMA rewrite is integrated into the operating system (thereby removing the existing limitations on single-compute node I/O performance) and/or until multiple readers are supported.

Guidelines and Limitations 4

Introduction

The chapter contains the following information:

- Memory usage information for the Paragon™ OSF/1 operating system.
- Guidelines for using the Paragon OSF/1 system software.
- List of the open bugs in Release 1.2 of the Paragon OSF/1 system software.
- List of the bugs fixed since Release 1.1 of the Paragon OSF/1 system software.

The lists of bugs are updated just before shipment and are also available online in the files */usr/share/release_notes/ss_buglist* and */usr/share/release_notes/ss_fixed* on the Paragon system.

Paragon™ OSF/1 Operating System Memory Usage

This section provides memory usage information for the Release 1.2 of the Paragon OSF/1 operating system. The total size of the Paragon OSF/1 operating system is 9.5M bytes on I/O and service nodes and 7.1M bytes on other nodes. This is the total for the microkernel and server. About half the Paragon OSF/1 operating system is resident at all times (it cannot be paged). The rest of the operating system must be resident to support system calls, but can be paged out.

Table 4-1. shows the memory usage for the components of the Paragon OSF/1 operating system.

Table 4-1. Paragon™ OSF/1 Operating System Memory Usage (1 of 2)

Component	Total	Description
microkernel[1]	3.1M bytes	The size of the assertionless and optimized microkernel is 876,160 bytes (text) + 113,824 bytes (data) + 510,688 bytes (bss) = 1,466,240 bytes. The microkernel also allocates memory at runtime (start-up memory). Wired page memory used by the microkernel is about 200 pages at 8K bytes per page. This consumes another 1.6M bytes (internal tables).
server[2] Runs on I/O and service nodes	6.0M bytes	The size of the server on I/O and service nodes is 1,583,040 bytes (text) + 116,864 bytes (data) + 375,392 bytes (bss) = 2,075,296 bytes. Server memory is virtual address space within the server task and can be paged to disk. The server has about 4M bytes allocated for internal tables, including file system buffers. In addition there are server symbols present in memory for debugging if the server is not stripped. This is about 250K bytes. In this release, the server is not stripped.
server.compute Runs on compute nodes, but not on I/O nodes	3.7M bytes	The size of the server on compute nodes is 1,210,592 bytes (text) + 97,952 bytes (data) + 352,567 bytes (bss) = 1,661,120 bytes. Server memory is virtual address space within the server task and can be paged to disk. The server has about 2M bytes allocated for internal tables.
emulator[3]	0.35M bytes	The emulator is linked in with each user program and shares virtual address space with it. The size of the assertionless, optimized emulator is 310,528 bytes (text) + 27,360 bytes (data) + 10,560 bytes (bss) = 348,448 bytes. The emulator resides in 512K bytes of the user virtual address space and can be paged to disk. This does not mean it is going to use the 512K bytes.

Table 4-1. Paragon™ OSF/1 Operating System Memory Usage (2 of 2)

Component	Total	Description
message buffers	1M bytes (default)	NX applications require message buffers in the user address space to support high-performance protocols. The default message buffer size for an application is 1M byte and can be overridden with application command line switches. The size must be larger for applications running on large numbers of nodes to provide adequate buffer space.
program overhead	0.33M bytes	Even a small NX program takes a significant amount of memory for all the libraries and standard data structures. A simple "hello world" program is 150,240 bytes (text) + 22,752 bytes (data) + 160,896 bytes (bss) = 333,888 bytes. An NX application takes at least 1.6M bytes of memory. This includes the default memory buffer, the emulator, and program overhead. On a 32M-byte node, the amount of memory available for an application without paging is 22.5M bytes with SPV running and 23.5M bytes without SPV running.

1. The microkernel is the only system software component that is in memory at all times (including its internal tables and buffers).
2. The server is in the memory initially, but as pages are needed any part of the server that is not used is paged out.
3. The emulator comes in with each process, and any part not used is paged out.

Using the Paragon™ System Software

Booting the Paragon™ System

We recommend booting your Paragon system with kernel assertions off.

Please contact SSD Customer Support for information about which Paragon OSF/1 kernel is best for the applications running on your system. See the online **reset** reference page for information about using the **reset** command to boot a Paragon system.

Recompile and Relink Application Code

When a new release of the system software is installed on your system, recompile and relink your application code using the compilers and system software libraries provided with the new release. You need to do this because executables from different releases are not compatible. Running applications compiled and linked with older software libraries (prior to Release R1.2) on the new Paragon OSF/1 system, the application will display the following message:

```
autoinit nx_initve: Error 216 occurred, unknown.
```

Gang Scheduling

Using gang scheduling will reduce the stability of a Paragon system at your site depending on the application mix you are running, the number of nodes that applications are running on, and how much paging occurs in an application. In general, small applications, which cause less paging, will be more stable than larger applications.

See the *Paragon™ System Administrator's Guide* for information on configuring partitions for gang scheduling.

Configuring Partitions For Gang Scheduling

The following describes a partition configuration that uses gang scheduling. This configuration has been characterized by running the system acceptance tests (SAT) on a 64-node system. In this configuration there are two scheduling layers:

- Layer 1 consists of a single instance of the SATs running in a subpartition that contains all the nodes of its parent partition (one 64-node subpartition).
- Layer 2 consists of four instances of the SATs, each running in their own subpartition of one quarter of the nodes of the parent partition (four nonoverlapping 16-node subpartitions).

The parent partition has read and write permissions, but no execute permissions. This allows subpartitions to be created in the parent partition, but applications cannot run directly in the parent partition. The subpartitions have execute permissions only.

In this configuration, the instance of the SATs running on all nodes alternates execution with the four instances of the SATs running on the subsets of nodes. Each of these two "layers" receive alternate execution slices for a duration specified by the rollin quantum. Here are the results:

- With the rollin quantum of the parent partition set to 5 minutes, the system SATs ran for 10 hours.
- With the rollin quantum of the parent partition set to 15 minutes, the system SATs ran for 12 hours.
- With the rollin quantum of the parent partition set to 30 minutes, the system SATs ran for 20 hours.

Higher Priority Applications Do Not Roll Out Lower Priority Applications

The allocator assigns partition layers to each gang-scheduled partition based on whether the application will fit in a layer; priority is not a consideration. If applications of different priorities are scheduled, you may find that not all higher priority applications will run at the same time, even if they can all fit in the same layer. For example, suppose you have a 16 node partition with the following applications to run:

- Application1 has priority of 10 and a size of 8 nodes.
- Application2 has priority of 5 and a size of 8 nodes.
- Application3 has priority of 10 and a size of 8 nodes.

Application1 and Application 2 are started first, then Application3 is started. Application3 is not allowed to run, because Application2 is already running and there is not enough space in the partition for Application3. Even though Application3 has a higher priority than Application2, it does not run before Application2.

SPV Not Started If Configuration Includes SUNMOS

The */sbin/init.d/spv* script has been modified to not start the *spvdaemon* if an alternate kernel is specified to execute on the compute nodes. The script prints the following error message if BOOT_ALT_NODE_LIST is not null:

```
Spv data collection NOT started: no support for alternate OSs.
```

Message Coprocessor

Before performing any of the installation steps, verify whether your system hardware has the message coprocessor (MCP) hardware. See "Verifying Required Hardware" on page 2-6 for information about the required hardware for the MCP.

The bootmagic string BOOT_MSG_PROC specifies whether to boot the system with system software that supports the MCP hardware. The bootmagic string BOOT_MSG_PROC specifies the following:

0 Boots the system with the system software for MCP hardware disabled. The system will run with any hardware configuration, but the MCP functionality is turned off.

1 Boots the system with the system software that supports the MCP hardware enabled. If there is only MCP hardware in the system, the system boots and runs properly. If there is MCP and non-MCP hardware in the system, the system does not boot.

NOTE

The bootmagic string BOOT_MSG_PROC is set automatically when the system is booted. There is no need to include this bootmagic string in the *MAGIC.MASTER* file.

Table 4-2. shows the effect of the bootmagic string BOOT_MSG_PROC on a system reset.

Table 4-2. Effect of BOOT_MSG_PROC on System Reset

BOOT_MSG_PROC	System Configuration	reset action
no value	All MCP boards.	BOOT_MSG_PROC=1 is put in the bootmagic file. The system boots properly.
no value	At least one board that is not an MCP board.	BOOT_MSG_PROC=0 is put in the bootmagic file. The system boots properly.
0	Any configuration.	The MAGIC.MASTER file takes precedence. The bootmagic string BOOT_MSG_PROC=0 is put in the bootmagic file.
1	All MCP boards.	The MAGIC.MASTER file takes precedence. The bootmagic string BOOT_MSG_PROC=1 is put in the bootmagic file.
1	At least one board that is not an MCP board.	Displays error message and exits. The system does not boot.

For a system that does not have MCP hardware and the bootmagic string BOOT_MSG_PROC is not in the *MAGIC.MASTER* file, a system reset will automatically disable the message coprocessor functionality in the system software by setting the bootmagic string BOOT_MSG_PROC as follows:

```
BOOT_MSG_PROC=0
```

If you add BOOT_MSG_PROC=0 to the *MAGIC.MASTER* file, the system will always disable the MCP functionality in the system software no matter what the hardware configuration is.

For a system that has all MCP hardware and the bootmagic string BOOT_MSG_PROC is not in the *MAGIC.MASTER* file, a system reset will automatically enable the message coprocessor functionality in the system software by setting the bootmagic string BOOT_MSG_PROC as follows:

```
BOOT_MSG_PROC=1
```

If you add BOOT_MSG_PROC=1 to the *MAGIC.MASTER* file, the system will always boot with the message coprocessor functionality in the system software enabled. Be aware that if the system has both MCP hardware and non-MCP hardware, the system will not boot and the following message will be displayed:

```
NOTICE FROM BOOTPP:
  The node found in Cabinet [x] BP [y] Slot [z] does not have the MCP ECO, but
  the bootmagic string BOOT_MSG_PROC is set to 1 in your MAGIC.MASTER file.
  You need to do one of the following:
    1. Replace the board with an ECO'd one OR
    2. Change your MAGIC.MASTER to contain BOOT_MSG_PROC=0 OR
    3. Remove the BOOT_MSG_PROC=1 entry from MAGIC.MASTER and the Message
       Copressor (MCP) will automatically be turned off during the
       boot process
Bootpp Exiting...
```

General Programming Guidelines

This software release does not require workarounds for most code that runs on less than 128 nodes. If you are running on a larger number of nodes or if you experience difficulty with your program, the following guidelines may be helpful.

1. Dynamically allocate memory at run time, especially if the application requires more than 10M bytes of memory. Use **malloc()** for C and **ALLOCATE** for Fortran. Typically, dynamic allocation gives a shorter execution time.

2. By keeping the load module as small as possible (under 23M bytes for 32M-byte nodes), you can prevent excessive paging in an application.

3. Maintain a maximum ratio of 32 compute nodes per physical device (MIO node).

4. Never use an **nx_initve()** or **nx_initve_rect()** system call in a program running in the compute partition.

Avoid Using Large Statically-Allocated Data Structures

Pages for statically-allocated data structures are touched at load time. This can substantially increase the load time for applications with large statically-allocated data structures.Use dynamically-allocated data structures when possible (for example, use **malloc()** for C or **ALLOCATE** for Fortran).

Global Operations

The global operations (**g...()** calls) are not tuned for maximum performance in the Release 1.2 system software. Only minor performance improvements can be expected over the Release 1.1 system software. A side effect is that the global operations may scale in unexpected ways. For example, the **gdsum()** call may exhibit a substantial linear component when purely logarithmic scaling might be expected.

SIGUSR2 Signal

The **SIGUSR2** signal value is not available to parallel applications because the system uses it to implement gang scheduling. This is true whether or not gang scheduling is used.

Message Passing

This section provides release information for Paragon OSF/1 message passing.

Global Sends

Global sends use a -1 value for the *node* parameter in a Paragon OSF/1 send system call. Global sends are now implemented using a tree-structured store-and-forward message passing strategy. This requires each node in an application to completely receive the message, otherwise, the next node in the tree will not receive the message. Use any of the Paragon OSF/1 receive calls to do the receive.

Synchronous Sends on Large Paragon™ Systems

An application that uses the **csend**() call for synchronous message passing can block or hang if you do not allocate enough memory for the application to handle messages. The memory allocated to a given node for message passing is based on the number of nodes in the application and the total memory available to the application for message passing. Therefore, as the number of nodes for an application increases, the memory available to each node for message passing decreases. With a large application running on a large Paragon system, this decrease can happen faster than expected causing the application to block. For example, an application that uses the **csend**() call for message passing may run fine on 64 nodes, but have problems running on 128 nodes, because there is not enough memory for message passing. The following workarounds can prevent this problem from happening:

- Post a receive before doing the synchronous send.
- Use a non-blocking send, for example, the **isend**() call.
- Decrease the message size for the application.
- Increase the memory buffer size for the application using the **-mbf** switch.
- Use the **-noc** switch to control the number of correspondents.

Setting the Process Type of a Controlling Process

A process can only change its process type with **setptype**() if its process type is **INVALID_PTYPE**. Once the process type is set to a value other than **INVALID_PTYPE**, it cannot be changed.

In an applications linked with the **-nx** switch, you cannot change the process type of a process. When you run an application that was linked with the **-nx** switch, the system sets the process type of all processes in the application to the value you specify with the **-pt** switch on the application command line (default 0). See the *Paragon™ User's Guide* for more information.

Do not call **setptype**() in a controlling process that does not do message passing, because this call assigns memory for message buffering that will be unused in this process.

Message Handlers

You must use **masktrap**() around any code in the main program that could interfere with calls in a handler established with one or more **h...**() calls.

Because it is often not obvious which calls could interfere with each other, use **masktrap**() to protect *all* library calls in the rest of the program that could call the same subsystems as the calls in the handler while the handler is active. For more information, see the discussion of **masktrap**() in the *Paragon™ User's Guide.*

Using the Allocator

With Release R1.2, you now specify how the allocator controls partitions and applications using the allocator configuration file */etc/nx/allocator.config*, instead of using command line switches. This functionality is described in the **allocator** and **allocator.config** manual pages in the *Paragon™ Commands Reference Manual.* You can also find information on this in the *Paragon™ System Administrator's Guide*. You should be particularly aware of the following changes to the allocator.

Space Sharing with the Allocator

Space sharing is now controlled with the *SPACE_SHARE* parameter in the allocator configuration file */etc/nx/allocator.config*, instead of using the allocator's **-tile** switch that was provided in previous releases. The description of the *SPACE_SHARE* parameter is in the **allocator** and **allocator.config** manual pages. See the online manual pages or the manual pages in the *Paragon™ Commands Reference Manual.*

Verifying MACS Accounts with the Allocator

Verifying MACS accounts is now controlled with the *USE_MACS* parameter in the allocator configuration file */etc/nx/allocator.config*, instead of using the allocator's **-MACS** switch that was provided in previous releases. The description of the *USE_MACS* parameter is included in the **allocator** and **allocator.config** online manual pages, but is not included in these manual pages in the *Paragon™ Commands Reference Manual.*

The following parameter can be specified in the file *allocator.config* to specify that the allocator verify a user's MACS account before running an application.

USE_MACS=boolean

Specifies whether the allocator must validate users' accounts with the Paragon Multi-User Accounting and Control System (MACS). This allows MACS to verify that users belong to valid MACS accounts. The *boolean* value specifies the following:

0 The allocator does not validate users' account IDs with MACS.

1 The allocator must validate users' account IDs with MACS.

The factory default for the *USE_MACS* parameter is 0 (this line is omitted from the *allocator.config* file); if this line is omitted or commented out or the *allocator.config* file is missing, the default value is 0.

This parameter is equivalent to the allocator switch **-MACS** used in previous releases. This parameter should be used instead of the **-MACS** switch.

I/O System

This section provides release information for the Paragon OSF/1 I/O system.

Maximum Compute Nodes Per I/O node

Applications should not attempt simultaneous I/O from greater than 32 compute nodes to any one I/O node (MIO) at one time. This is true for UFS, NFS, and PFS (striped on one I/O node). This is due to underlying system limitations that cause low bandwidth and/or system hangs when multiple clients are accessing that same I/O node. The system administrator must be careful to configure the PFS file systems so that this restriction is enforced across multiple simultaneous applications. For more information about configuring PFS file systems, see the *Paragon™ System Administrator's Guide*. Users must understand PFS I/O modes and stripe attributes in order to code their applications within this restriction. For more information about programming PFS applications, see the *Paragon™ User's Guide*.

I/O Request Size Limitation

Applications should avoid simultaneous large I/O requests to any single I/O node (MIO). This may cause system performance to degrade and may cause the system to hang. This is due to excessive paging when the total size of the I/O transfers exceeds the physical memory size of the I/O node. A suggested workaround is to break large I/O requests into smaller I/O requests, so the smaller I/O requests will fit into physical memory.

For example, if a 64-node application issues simultaneous requests for 512K-byte I/O from each compute node using the **M_RECORD** I/O mode, a system with two I/O nodes would need up to 16M bytes of buffer space for each I/O node. A system with 16M-byte nodes has approximately 10M bytes available memory, while a system with 32M-byte I/O nodes has approximately 26M bytes available.

A workaround is to break your I/O requests into smaller blocks, or reconfigure your PFS to be striped across additional I/O nodes.

Logical Volume Manager

Although documented in the OSF/1 manuals, the logical volume manager (LVM) is inappropriate for a Paragon OSF/1 system. The LVM software and online manual pages have been removed from the system. The LVM software is a set of resources that OSF/1 provides for managing disk storage in a system. The following LVM commands are not installed on the Paragon OSF/1 system:

lvchange	**lvremove**	**pvmove**	**vgreduce**
lvcreate	**lvsync**	**vgchange**	**vgremove**
lvdisplay	**pvchange**	**vgcreate**	**vgsync**
lvextend	**pvcreate**	**vgdisplay**	
lvreduce	**pvdisplay**	**vgextend**	

Verifying fsck Results

When you boot the system, the **fsck** command is automatically run on each file system defined in */etc/fstab* (such as */home*, */pfs* and the PFS stripe directories typically mounted on */home/.sdirs/vol**).

In the case of a severely damaged file system, the **fsck** will fail and a message will be displayed to the screen. The message will scroll off the screen before the reboot is complete and you may not notice it. The system will continue to boot to a multiuser state with no other indication that anything is wrong.

If the **fsck** fails, the affected file system will not be mounted. Any files written to the unmounted file system are written into the root file system, not the desired file system. Later, if you notice that a file system did not mount and you **fsck** and mount it, the files that were written into the root file system become hidden. To avoid this problem, after each boot you should manually compare the mounted file systems with the contents of the *fstab* file. Make sure that the PFS stripe directories are mounted or the **fsck** data will go into the */home* file system.

HIPPI Limitations

When the Paragon is booting, the boot process creates the file */usr/tmp/raw_hippi.log* and prints a message indicating that it is configuring HIPPI boards. This happens even if the system has no HIPPI boards, in which case, you can delete the log file and ignore the message. If the system does have a HIPPI board, refer to the *Paragon™ High-Performance Parallel Interface Manual* and the **set_hippi_buffers** reference page for additional information.

The optimal number of bytes to write for a **hippi_write**() call from a compute node is 256K bytes. Larger sizes will have a small performance degradation.

The **hippi_bind**() call does not use the port parameter. The call ignores values given for this parameter.

For Fab 3 HIPPI boards, the HIPPI console reports that the ULA is invalid. To obtain the correct ULA, look on the HIPPI controller for the ULA label.

Use the bootmagic string TCP_SPACE_SIZE with caution. A value larger than 64K (65536) can cause the system to panic. Use this bootmagic string at your own risk.

System Administration

This section provides release information for Paragon system administration.

The reset Script

For information on the **reset** script, see the **reset(8)** manual page either online or in the *Paragon™ Commands Reference Manual.*

Shutting Down the Paragon™ System

Use the following command sequence to reboot the system. In a console window:

```
# cd /
# sync;sync;sync
# shutdown now
# umount -A
# halt
```

This writes the system buffers, kills all running processes, brings the system to single-user mode, unmounts the file systems, and halts the system. You can safely power down the system after this sequence.

To reboot the system, use the following command after the **halt** command completes.

```
DS# reset
```

For more information about system shutdown, see the *Paragon™ System Administrator's Guide.*

Backing Up a File System to the DAT Tape Drive

The following example allows you to efficiently use the **dump** command to back up a file to the DAT tape drive on the Paragon system:

```
/usr/sbin/dump -f /dev/io0/rmt6 -c -d 61000 -S 11000 /dev/io0/rrz0a
```

The **-f** switch specifies using the */dev/io0/rmt6* storage device for the dump. The **-c** switch specifies that the dump medium is a cartridge tape. The **-d** switch specifies a write density of 61000 bits per inch (bpi) for the storage medium. The **-S** switch specifies the size of the dump tape to be 11000 feet. Using these values the command dumps the file system on */dev/io0/rrz0a* to the DAT drive.

Support for the Fast DAT Tape Drive

Release R1.2 supports the Hewlett-Packard Model C1533A DAT tape drive. The Model C1533A DAT supports data compression and uses both the 90m DAT tape and the 120m DAT tape. The 120M DAT tape is a 4-mm tape that holds up to 4G bytes of data. The 90m and 120m tapes have been tested for doing backups. The older model DAT, the Hewlett-Packard Model 35470A DAT tape drive, does not support data compression and cannot use the 120M DAT tape. The device name for the Model C1533A DAT tape drive is */dev/io0/rmt6* for uncompressed data and */dev/io0/rmtc6* for compressed data. Please contact SSD Customer Support for information on using the Hewlett-Packard Model C1533A DAT tape drive.

Shared Virtual Memory

The OSF/1 operating system supports a feature called *shared virtual memory* that can be used to share data between processes. Both the System V shared-memory calls and the mapped-memory interface are supported (see **shmget(2)** and **mmap(2)** in the *OSF/1 Programmer's Reference* for information about these calls). However, these calls can only be used to share memory between processes running on the *same node*. Any attempt to share memory between processes running on different nodes will fail.

NQS For Workstations

The Paragon NQS Network Queueing System (NQS) supports a networked environment. However, the NQS executable files shipped with the Paragon OSF/1 system software are *only* for Paragon systems. If you want to use NQS from your workstation, you have to separately obtain the NQS software for your workstation.

NFS For Workstations

The diagnostic station OS installation instructions do not currently point out that SCO UNIX requires that a "key" be entered during software installation. This is an activation key. If you use the same source media (floppies/tape) to load the SCO UNIX on another system, and then try to use NFS, the system detects that another system is on the network with the same activation key, and then NFS is "disabled" and refuses to work.

Bug Lists for the Paragon™ System Software

This section contains the following bug lists for Release 1.2 of the Paragon system software:

- Open bug list.
- Fixed bug list.

The open bug list lists the open bugs Release 1.2 of the Paragon system software. The open bug list is organized in alphabetical order by subsystem name. The bug listing includes the following:

- Bug number.
- Subsystem name.
- Bug synopsis.
- Bug description.

The fixed bug list lists the bugs fixed since Release 1.1 of the Paragon system software and included in Release 1.2. The fixed bug list is organized in numerical order by bug number. The bug listing includes the following:

- Bug number.
- Subsystem name.
- Bug synopsis.

The lists of bugs are updated just before shipment and are also available online in the files */usr/share/release_notes/ss_buglist* and */usr/share/release_notes/ss_fixed* on the Paragon system.

Open Bug List

The following lists the open bugs for Release 1.2 of the Paragon system software:

5355 BOOT PROCESS — If scanio is used, output pending on the console can hang system

If the Paragon is booted using scanio (BOOT_CONSOLE=s), and the console has been subsequently detached (with ~.), if output comes to the console, the whole system will hang up waiting for someone to read that output. When hung, the system will not even respond to a "ping".

To bring the system back, just re-attach to the console and receive the message.

7282 BOOT PROCESS — fscan gets behind in echoing characters

When using the fscan interface, the system sometimes gets behind in echoing characters on the terminal screen. For example, if you type the command "who <RETURN>", the "wh" is displayed before the carriage return and the "o" after the return. A workaround for this problem is to type the following on the FSCAN> command line:

```
FSCAN>  calibrate <RETURN>
FSCAN>  adjust <RETURN>
```

This operation resets fscan without a reboot.

7954 BOOT PROCESS — When booting, problems detected with fsck are sometimes handled improperly

If fsck encounters unrepairable file system problems during the boot process, the system is suppose to leave you in single-user mode so you can repair the file system before you allow users on the system. If the file system is in bad shape (i.e., its superblock is corrupted), the boot process may continue to multiuser mode (with the errors detected by fsck scrolling off the screen) instead of going to single-user mode. If you experience this problem, try to bring the system up in single-user mode and run fsck manually. If fsck is not able to clean up the file system, you may need o rebuild the partition using newfs.

8330 BOOT PROCESS — Watchdog incorrectly says "Node not responding"

Sometimes when fscan says a node is not responding, it is actually bogged down and quickly recovers. No reboot is necessary. To eliminate incorrect messages like "Node not responding", you can turn off notify, as follows:

```
FSCAN> set notify off
```

8436 BOOT PROCESS — fscan interface sometimes sends garbled output to the console device

When using commands like ps and ls, fscan sometimes sends incoherent output to the console. If you are doing a lot of text-related work and garbled output becomes a problem, rlogin to your Paragon system directly, instead of using the console interface.

9110 BOOT PROCESS — fscan is reporting incorrect "Node xx (cbs=xx) is in the DEBUGGER!" message

The fscan program occasionally outputs incorrect "Node xx (cbs=xx) is in the DEBUGGER!" messages on GP nodes. To eliminate these incorrect messages, you can turn off notify, as follows:

```
FSCAN> set notify off
```

6179 HIPPI — hippi_bind() doesn't use the port parameter to set the filter

The hippi_bind() function is not currently using its "port" parameter to set the filter as documented.

6495 HIPPI — no way to report ULA on HIPPI controller from user level

There is currently no easy way to determine the ULA of a HIPPI controller. Using the "arp -a" command fails with the error message "arp: could not allocate 0 arptab entries".

You can determine the ULA by looking on the controller card for its ULA label, or by using "kt" on the HIPPI node and looking at the kernel boot output.

5029 LIBNX — The led() system call has no effect on the LED displays

8671 LIBNX — The _hsendrecv() libnx function doesn't return a valid value

The _hsendrecv() function should return a 0 on correct operation and a -1 on error (with the errno set to the appropriate error message). Currently, the function does not return any useful information.

8791 LIBNX BETA: global csend() does not work with hrecv()

The handler interface between csend() and hrecv() does not work properly when sending messages globally (csend(-1)), resulting in the message information being incorrectly received. The workaround is to use csend() only to send messages directly to nodes, rather than globally.

7937 MACS MACS database-dependent utilities don't handle "acctonly" mode correctly

The MACS database-dependent utilities (such as macupdate) are intended for use in the "macdmode", not in the "acctonly" mode. If you use these utilities in the acctonly mode, you will get an error message; however, the error message will not indicate that you are in the wrong mode.

8753 MACS MACS doesn't notice when it can no longer write logfiles

If the /var file system fills up, preventing the writing of future MACS logs, the system administrator is not notified. To avoid loosing log data, check /var periodically and remove old log data so that the /var partition does not fill up unnoticed.

9226 MACS Due to MACS startup script errors, SI may not produce the correct log entry

A false boot record may get written due to the way the start script tries to determine if a reboot just happened. This problem occurs the day after a reboot when MACS is stopped and started but the system is not rebooted. There is no workaround for this problem.

9273 MACS BETA: R1.2 MACS commands cannot process R1.1 log files

The only workaround for this problem is to install an R1.1 version of MACS to use in reading old log files.

9279 MACS The argument to the -p switch in jrec requires a final "/"

The argument to the -p switch in jrec is a pathname. Jrec joins this pathname to file names found in the directory to access the files. When using the -p argument, you must add a "/" to the end of the pathname for the argument to be handled correctly.

9445 MACS BETA:/etc/nxaccount requires world readable perms w/ macs dflt accnts

You must make the MACS file /etc/nxaccount world readable for all MACS default accounts to use the MACS administration functions.

7300 MESH UTILS Allocator doesn't detect typos or bogus values in allocator.config file

The allocator does not detect typing errors and incorrect values in the /etc/nx/allocator.config file and/or generate error messages to indicate problems. So, for example, if you enter a value of "20v" for MIN_RQ_ALLOWED instead of "20m", the allocator will ignore the parameter and not set a limit. (It does detect incorrectly spelled variable names.) There is no workaround for this problem other than to double-check entries in the allocator.config file.

7624 MESH UTILS SMD returns inconsistent timing values to MACS

The CPU time reported by SMD can be significantly larger than the correct time, which is the (wall clock time) * (number of nodes). There is not workaround for this problem if it occurs.

7843 MESH UTILS mkpart -sz 36/application -sz 36 does not allocate a square

A mkpart -sz 36 does not allocate a square, even though there is room for a square of that size. There is no workaround for this problem.

8284 MESH UTILS Higher priority does NOT roll out lower priority job in active layer

Jobs are assigned to layers based on whether the job will fit in the layer. Priority is not a consideration. Here is an example of the problems in a 16 node partition:

```
Job #1: priority 10 size 8
Job #2: priority 5 size 8
Job #3: priority 10 size 8
```

Jobs #1 and #2 are issued first, followed by job #3. Since job #3 is high priority than job #2, job #3 should preempt job #2, but it is not allowed to. Job #3 must wait until job #1 finishes. There is no workaround for this problem other than to manually schedule jobs in the order you wish them worked on.

8432 MESH UTILS — Using nx_pri() to lower job priority does NOT roll job out for other ovlp jobs

When two jobs of different priorities are run in an overlapping partition and nx_pri() is used to lower the priority of the higher priority job, the job is not rolled out when its priority becomes lower than the other job. There is no workaround for this problem.

8464 MESH UTILS — chpart -rq $RQ <SPS_part> may cause system instability

Using the command "chpart -rq $RQ <SPS_part>" can cause the REJECT_PLK=1 control in the /etc/nx/allocator.config file to be bypassed, resulting in system instability.

3168 MESSAGE PASSING — msgcancel does not work on merged message IDs.

Using msgcancel() on a message ID that is the result of a msgmerge() does not always cancel all of the merged messages.

3353 MESSAGE PASSING — Invalid info parameter pointer to _irecvx or _crecvx causes core dump.

Calling _irecvx() or _crecvx() with an invalid info parameter pointer produces no error and terminates the process.

3710 MESSAGE PASSING — flushmsg() is not implemented

The flushmsg() call is not implemented in the current release.

4302 MESSAGE PASSING — Global sends can hang the application when there are > 1 process per node.

If an application has more than one process per node, and more than one process on one node sends or receives a message to the same other node (for example, if one application has two processes on node A and both of them either send a message to node B or receive a message from node B), the application may hang.

4329 MESSAGE PASSING — autoinit program execs (no fork) and child can't message pass.

If a compute node program execs a new image without first forking a child, the new program fails when it attempts to pass messages even if it does a setptype. The error message displayed is:
"Invalid ccode pointer"

4700 MESSAGE PASSING Global send to different ptype may work or not, depending on sending node.

If a process sends to node -1 (all nodes) with a process type different than its own process type, it may succeed or may hang. Whether or not it succeeds seems to depend on the node number of the sending node.

4703 MESSAGE PASSING Repeatedly nx_nfork()ing children crashes machine with assertion in mcmsg_inq.c

Under some circumstances, creating very large numbers of child processes by repeatedly calling nx_nfork() in a single application crashes the system with the error message "Assertion failed: file ../../../../../src/mk/kernel/i860ipsc/mcmsg/mcmsg_inq.c, line 548". Occasionally the machine hangs with no assertion also. The number of child processes that can be created before the problem is seen depends on the process types of the child processes and whether or not they do message passing, but the problem does not appear unless at least 600 child processes are created.

4788 MESSAGE PASSING In order for a global send to complete each node must officially receive it.

Global sends (send to node -1) are currently implemented using a tree-structured store-and-forward strategy. This implementation is much faster than the previous strategy, but it requires that each node in the tree must completely receive the message (by calling crecv() or irecv()/msgwait()) before it can pass it on. This means that if one node does not receive the message, other nodes (those "further down the tree") will not receive the message until the first node has received it.

4886 MESSAGE PASSING msgcancel doesn't appear to cancel a message id.

Calling msgcancel() does not currently release the specified message ID. After a call to msgcancel(), the number of available asynchronous message IDs is not increased.

5683 MESSAGE PASSING Call fork() then execve() from child process in compute caused assertion failed.

If a process is created in the compute partition as a child process of a controlling process in the service partition, and the child process calls fork() and execve(), it hangs with the following error message:

```
Pid <n> assertion failed
../../../../../../src/usr/ccs/lib/libnx/rklib.c line 587
```

5739 MESSAGE PASSING — Global sends to other ptypes do not reach all of the intended processes.

A global send to another ptype does not reach the target process on the sending node. For example, suppose an application has two processes per node, one with process type A and the other with process type B. If a process with process type A sends to process type B on node -1 (all nodes), the process with process type B on the node that sent the message does not receive it.

If the sending call is an hsend(), occasionally NO message is received by ANY process.

5838 MESSAGE PASSING — After an "exec", compute node no longer recv's messages from controlling process

If a controlling process forks itself onto compute nodes (using nx_nfork()) and the child process calls execvp() to execute a new program, the newly exec'ed program does not receive messages sent by the controlling process, even if it calls setptype(0).

6216 MESSAGE PASSING — NX message passing involving .service does not work

When a message-passing program is run in the service partition, it does not work: the values returned by numnodes() may be invalid, and the program may hang.

7647 MESSAGE PASSING — msgmerge(mid,mid) causes subsequent msgdone() to hang

Calling msgmerge(mid,mid), where you specify a valid mid twice, causes a subsequent msgdone() to hang if a matching message is pending. To avoid this problem, don't call msgmerge(mid1, mid2) where mid1 and mid2 are identical.

8227 MESSAGE PASSING — Posting continuous global, asynchronous sends can hang the system.

A loop that continuously posts global asynchronous sends (using _isend() to send to node -1) without having receiving nodes post irecv()'s periodically to release posted Message IDs can cause the system to hang. This condition is caused by the program using up the finite number of available Message IDs. When this happens, _isend() returns a -1 (isend() error: to many requests"). When the program then calls exit() in response to the error, the system hangs instead of terminating the program. There is no workaround for this problem. However, you may be able to avert this problem by restructuring your program to have nodes receive messages more frequently to free posted Message IDs.

8358 MESSAGE PASSING — Output is missing/corrupted/doubled when printing from hrecv() handler

This problem results from the print() functions being called from two different threads running in the same process. Many of the functions in libc (including printf()) are not thread safe. Invoking a handler with hrecv() starts a new thread for the handler. Then, if this thread and another thread call the printf() function at the same time, the resulting output can have missing, corrupted, and/or double characters.

8439 MESSAGE PASSING — Using -plk with a large process may hang the node

If a process that is larger than the available memory space on a node locks memory using -plk, the node may hang. Use -plk only when the process that is locking memory requires less space than the available memory on the node.

6168 MISCELLANEOUS — BADNODES.TXT is not updated with failed nodes nor accessed by bootpp

When a node is detected as bad, and has been the cause of 'n' successive reboots, the watchdog should add this node to a file called BADNODES.TXT, and "bootpp" should read this file and remove any node from the list of valid nodes. This would prevent the watchdog from booting a machine over and over again. However, this file is currently not created by the watchdog and "bootpp" does not use it.

5740 NFS — Unable to NFS mount VAX/VMS-resident filesystems

Attempting to NFS-mount a filesystem from a VAX/VMS disk causes a trap. The VAX believes that the filesystem has been properly mounted on the Paragon.

6644 NFS — NFS mount to Convex Unitree fails, or gives weird results

Attempting an NFS mount of a Convex Unitree file system on the Paragon system does not work. It may time out with an error message to the effect that the remote server is not responding, or may appear to succeed but give unexpected results (such as erroneous and variable permissions, ownership, and modification dates on the mount point).

7618 NFS — Cannot export file systems from the second ethernet

You cannot export file systems through a Paragon's second ethernet connection. This means that a Paragon cannot operate as a file server over a second ethernet. All file exports go through the boot node. There is currently no workaround for this problem.

7662 NFS — NFS mounts frequently hang at boot time when the netserver is not on boot node

If the network configuration at boot time consists of only bringing up non-boot node netservers, immediate attempts to NFS mount remote file systems will frequently hang the system. Since the hang occurs at boot time, there is no way to break out of the hang without rebooting the system. A workaround that sometimes solves this problem is to add a sleep call of 5 to 7 seconds to the /sbin/init.d/inet initialization file right after the default routes are established. If the system still hangs with the sleep delay, it generally comes up OK on a reboot.

9098 NQS — NQS dies quietly because user partitions overlap NQS nodes

When setting up NQS node sets, the system administrator should set the permissions on the nodes to disallow other users from including them in their partitions. If this is not done and a user submits a job to a node_set that includes NQS nodes, NQS will die quietly: messages indicating the problem are written to the log file, but not to the screen.

9184 NQS — BETA: NQS dies when wrong path for a pipe queue is used

If you create a pipe queue using a wrong path name to the client, the following three things happen. (1) The NQS qmgr does not detect the problem. (2) If a job is submitted to the (not installed) pipe queue, NQS dies without the user being informed. (3) You must setup NQS again from scratch to get a working configuration again.

6276 OSF COMMANDS — vmstat CPU information fields always 0

The values of the "cpu" fields in the output of "vmstat" are currently always 0.

6509 OSF COMMANDS There is no way of killing the amd process started on a I/O node in .compute

If you start up an "amd" process on an I/O node which is in the compute partition, there is no way of knowing whether it got started or not. The problem is that "ps" cannot examine nodes in partitions other than the service partition.

A side effect of this problem is that there is no way to kill such a process, since there is no way to find out its PID.

6798 OSF COMMANDS RPM/bus test piped to more logs out window, produces <defunct> process?

If you run the "rpm/bus" test (which must be run by root) and pipe the output into "more", your root shell may be unexpectedly suspended or terminated, and a "ps" or "pspart" may show "<defunct>" processes.

7371 OSF COMMANDS dump doesn't work with /etc/dumpdates

The dump command will back up a single file system if the file system is specified on the command line, but it will not back up multiple file systems specified in /etc/dumpdates. If you attempt to back up multiple file systems in this manner, the command will hang, give you an error message, or return right away without doing anything, depending on what is in the /etc/dumpdates file. To backup multiple file systems, you can use consecutive dump commands, with the -N option to avoid rewriting while each file system is getting backed up.

7563 OSF COMMANDS Transitions between run levels are not clean

Using the init command to go from multiuser mode to single user mode and back again to multiuser mode does not work cleanly. To go back to multiuser mode reliably, you will need to reboot the system.

8778 OSF COMMANDS fsck hangs during reset if reset occurred while devices were being formatted

If the system is reset while one or more RAID devices are being formatted, fsck will hang during the reboot process. The solution to this problem is to go into single-user mode and edit /etc/fstab to remove references to the devices that were being formatted when the system was reset. Then, reboot the system and reformat the devices. After reformatting the devices, you can partition the devices and add entries to /etc/fstab again.

9377 OSF COMMANDS	invoking rlogin deamon with -l option does not work
6409 OSF LIBS	National Language Support locale categories LC_COLLATE & LC_CTYPE no longer work

If you use setlocale() to change the value of LC_COLLATE or LC_CTYPE to a language other than the default, the calls tolower(), toupper(), strcoll() and strxfrm() return incorrect values.

7232 OSF LIBS	sigwait() leaves process sig mask blocked when called concurrently
8561 OSF LIBS	Floating-point accuracy of printf() and other functions is not IEEE compliant

The floating point accuracy of atof(), _dsto2fp(), fcvt(), and ecvt() are not up to IEEE standards when converting double-precision numbers. These functions in turn affect printf() and scanf(). For example, when converting double-precision numbers using printf() or scanf(), the last couple of digits (least-significant) may not be correct. This is a software problem, for which there is currently no workaround.

8679 OSF LIBS	libc.a does not contains the functions fgetpwent() and fgetgrent()
6733 OSF MICROKERNEL	code executes with -Mnoxp, fails with -Mxp

Certain programs that exhibit unexpected behavior during asynchronous message passing when compiled with the default compiler flag -Mxp work as expected when compiled with -Mnoxp. The problem appears to be associated with the use of vectorized loops; if this happens to you, try recompiling any code that uses vectorized loops with -Mnoxp.

6759 OSF MICROKERNEL	multiple ^C when loading // apps hangs window, sometimes hangs system

If you attempt to interrupt a parallel application as it is loading by repeatedly pressing <Ctrl-C>, you can hang the session and may hang the entire system. Note that pressing a single <Ctrl-C> works just fine; the problem only occurs if you press several <Ctrl-C>'s.

6832 OSF MICROKERNEL	Writing PFS file from 64 node in M_RECORD mode caused kernel panic

Under some circumstances, writing large amounts of data to a PFS file system in I/O mode M_RECORD can cause the kernel to panic. One test case that shows the problem causes the panic by writing from 64 compute nodes to 1 I/O node using a 64K-byte request size. (Note that it is strongly encouraged that PFS applications use 32 or fewer compute nodes per I/O node.)

6837 OSF MICROKERNEL Exceeding "safe" limits of 32 nodes per I/O node on PFS causes hangs/crashes

In the current release, it is strongly recommended that you do not perform PFS I/O when the number of compute nodes involved is more than 32 times the number of I/O nodes involved. If you exceed this limit -- especially if the ratio of compute nodes to I/O nodes is 64:1 or greater -- the system is likely to hang or crash.

8361 OSF MICROKERNEL Reading or writing large blocks of data may panic the system

The cause of this problem is not known. If the system panics while you are reading or writing large blocks of data, try reducing the size of the blocks.

8433 OSF MICROKERNEL BETA: Reproducible (pmap_expand) on compute node

This panic occurs when running programs on two different partitions with different priorities (such as epl 5 and epl 9). Some previous occurrences of this problem have been traced to a cache coherency problem. The problem was fixed with a board replacement.

8480 OSF MICROKERNEL BETA: Assertion failed when running 'sat -c random comtest mxm mplinpack'

This system hang was caused by a port leak that resulted from heavy paging by the node. Any process that pages heavily may eventually hang the node for this reason. There is currently no workaround for this problem.

8509 OSF MICROKERNEL Random Kernel panic (zone_checkit+0x5c: ld.l) running sched tests

This panic is rare. It sometimes occurs in when terminating scheduled processes. Terminating a process or program that is inactive (i.e., rolled out") using the kill(1) or rmpart -f commands can result in this panic. There is no know workaround for this problem. (Do not hesitate to use kill or rmpart.)

8616 OSF MICROKERNEL — Heavy paging in an I/O node can cause the node to hang; free page=29, able_rec=0

If the compute-to-I/O node ratio in a system exceeds approximately 32:1 and heavy paging to an I/O node occurs (due to a Mach microkernel memory starvation problem) the I/O node can potentially hang. This problem can also occur if the compute-to-I/O node ratio is very large and the highly parallel M_RECORD file sharing mode is used to access a PFS file simultaneously from many compute nodes. Depending on the number of nodes and the application's read()/write() request size, this may far exceed the I/O node's ability to buffer incoming or outgoing PFS file data, resulting in heavy paging and possibly the memory starvation hang.

9244 OSF MICROKERNEL — Programs with large static .bss may require a long time to load

If your program has a very large static segment (.bss), the program may take quite a while to load and, in some cases, cause you to think it is not loading. This problem results from the system touching every page in the .bss segment when it is being loaded. A workaround for this problem is to change the large statically allocated data structures in your problem to dynamic allocation (malloc()).

4708 OSF SERVERS — Mount of two partitions on same directory permitted

It is currently possible to mount more than one disk partition on the same directory. The second partition mounted "masks" the first (that is, the contents of the mount point appear to be the contents of the second partition mounted). Once this has occurred, attempting to unmount one of these partitions may actually unmount the other.

4974 OSF SERVERS — kill -9 of 140-node parallel app. that has been ^Z'd hangs

Under some circumstances, suspending a large parallel application (with <Ctrl-Z> and then killing the suspended application (with "kill -9") can hang the entire login session.

7460 OSF SERVERS — can't run lint on server/emulator

7923 OSF SERVERS — Multiple concurrent opens can slow down server such that it appears hung

Any application program that concurrently opens a large number of files in a PFS or UFS file system can cause the system to slow down so much that it appears to be hung. There is no workaround for this problem other than to limit the number of concurrent opens that an application conducts.

7934 OSF SERVERS hostname or settime commands hang system during single-user mode

In single-user mode, the hostname and settime commands hang. These commands only work after bootmesh has run.

8398 OSF SERVERS System crashes when FORTRAN open() is called from hundreds of nodes

Opening the same file from hundreds of nodes in a FORTRAN program using an open() call can cause a performance bottleneck, which can in turn result in a system crash. A possible workaround is to remove any synchronization functions (such as gsync() calls) from the program that precede the open() call.

8400 OSF SERVERS The server does not provide support for "quota"

The R1.2 Paragon server is build without "QUOTA" being defined. As a result, none of the quota related utilities (such as quota(), quotactl(), edquota(), and quotaon()) in cmds/libs are supported. There is no workaround for this problem.

8720 OSF SERVERS OSF accounting (acct) is not usable with Paragon systems

The standard Unix OSF accounting functions under /usr/lib/acct do not function well on Paragon systems. When using these functions, some of the data that is collected is incomplete, due to incompatibility with the multinode environment. There is no workaround for this problem.

8732 OSF SERVERS Server does not print all critical warnings to console

Some O/S errors are only printed to the local node, not to the console. These messages can only seen if a sysadmin explicitly uses fscan to scan over to the node where the error occurred.

8786 OSF SERVERS Wired page crash (due to heavy UFS usage)

Wiring down all the memory on a node can over time cause a machine crash. One example of this problem occurred when an application attempted to write to a common Unix (UFS) file simultaneously from hundreds of nodes. This operation resulted in all the pages of a boot node being wired down. One solutions to this particular problem is to not attempt to write an UFS file simultaneously from a large number of nodes.

8796 OSF SERVERS Wired page crash (due to MACS operation)

Wiring down all the memory on a boot node can cause the system to hang. One example of this problem occurred when a MACS operation caused the /var partition to overflow with data. There is no workaround for this problem other than avoid operations that wire down all the memory on a node.

8974 OSF SERVERS Making many fork() calls on a node can crash the system

Running a program that makes a lot of fork() calls on a node while the node is low on available memory can result in a system panic.

8985 OSF SERVERS Panic: server_thread_deregister while quitting IPD

Sending a SIGALRM signal to a process stopped at a breakpoint under IPD may cause the system to panic upon the exit of IPD.

9116 OSF SERVERS A pthread can not install a process wide handler routine for SIGFPE.

A pthread can not install a process-wide handler routine for SIGFPE as other signals can. For example, a pthread can install a handler routine to catch the exception SIGSEGV or SIGBUS. Then any pthread that causes a SIGSEGV or SIGBUS signal will invoke the handler routine.

9284 OSF SERVERS Pressing CTRL-C before profil() turns off can cause subsequent hang system hang

Interrupting a program that calls profil() with CTRL-C may result in the program not exiting and hanging the system. The program where this bug was observed called profil() to turn on the profil function then called profil() again to turn off the function. When a CTRL-C was pressed before profil was turned off and program execution was resumed, the system hung on the profil() call to turn off the function.

9323 OSF SERVERS BETA: shutdown commands don't always bring down the system cleanly

The commands "shutdown", "init 0", "fastboot", "halt", "fasthalt", "reboot", and other OSF/1 shutdown commands do not always shut the system down cleanly. To be sure of a proper shutdown, always use the following sequence of commands:

```
# sync;sync;sync
# shutdown now
# umount -A
# halt
```

This sequence of commands is documented in the Paragon System Administrator's Guide.

9345 OSF SERVERS — Calling nx_initve() twice may cause a panic

Calling nx_initve() twice within a single application can cause a kernel assertion/panic. This error can occur either explicitly, when calling nx_initve() more than once in the application, or implicitly, when an application compiled with the -nx switch calls nx_initve(). In the latter case nx_initve() is called automatically during application startup and a subsequent call to it can cause the assertion/panic.

9373 OSF SERVERS — mount allows a device to be mounted on a directory that is busy

The server lets you mount a device to a directory that is busy, instead of prohibiting this action and returning an error message. If you mount a device to a busy directory, the directory will not be affected. When you unmount the device, the directory will be returned to its original state.

6396 OSF SYSTEM CALLS — When poll() is interrupted by child signal, parent hangs until child exits

9138 PAGING TREES — "PAGE_TO rz0c" in DEVCONF.TXT handled improperly

Placing a PAGE_TO rz0x list in DEVCONF.TXT in an attempt to construct a paging tree in RAID partitions does not work as expected. A workaround for this problem is to enter the PAGE_TO list in MAGIC.MASTER manually, instead of placing it in DEVCONF.TXT.

9139 PAGING TREES — Double colon incorrectly placed in PAGER_NODE list for auto paging tree

When attempting to generate an automatic paging tree using the -P1 flag for bootpp in the "reset" script, an error may occur on reset. This error may be the result of a double colon (::) that is incorrectly generated in the PAGER_NODE line of bootmagic. To avoid this problem, enter the PAGER_NODE line in MAGIC.MASTER manually and avoid using "-P1" to generate an automatic paging tree.

6237 PFS — gopen() allows opening of device special files, inconsistent with User's Guide

According to the Paragon User's Guide, gopen() can only be used to open ordinary files. However, if you gopen() a device such as a tape or TTY, the gopen() succeeds (it should fail). Closing the resulting file descriptor may hang; the hung application can be terminated with <Ctrl-C>.

The setiomode() call also does not give any error if used on a TTY (such as stdin, stdout or stderr).

6965 PFS — FORTRAN formatted I/O functions do not work with PFS I/O modes

Using the FORTRAN formatted I/O functions (such as write and read) in conjunction with PFS I/O modes can lead to program exits and system hangs. For example, using a combination of setiomode() and a FORTRAN write statement can cause a system hang. There is no workaround for this problem.

4507 PTHREADS — Can't pass messages between different pthreads in the same process

If a process has multiple threads, the threads may not be able to exchange messages with each other. The message appears to be received properly, but the received data is sometimes scrambled or invalid.

5895 PTHREADS — Under heavy load the call pthread_cond_timedwait("1 sec wait") never returns

Under heavy load, when a node is experiencing significant paging, the system call pthread_cond_timedwait() with a 1 second timeout value sometimes never returns.

7237 PTHREADS — exit() and wait() are not pthread safe

The reentrant C library functions exit() and wait()are not thread safe. For example, a race condition between wait() returning and exit() being called, each by a thread within a multi-threaded process, sometimes causes the process to hang. One workaround for this problem is to use a mutex to keep exit() from being called while calling wait(). Another is to use a waitpid(-1,&status,WNOHANG) call in a loop with a mutex around it and the exit() call.

7320 PTHREADS The libm.a function are not safe to cancel

The pthread_setasynccancel() man page incorrectly lists the libm.a functions safe to cancel. They are not, because libm.a functions are not thread safe.

7468 PTHREADS pthread_setcancel sometimes fails

When general cancellation of pthreads is blocked by pthread_setcancel(), pthreads are sometimes cancelled anyway, when functions that set up "cancellation points" are used. The pthread_cond_wait(), pthread_join(), and pthread_cond_timedwait() functions (which are "cancellation points") incorrectly carry out cancellation requests even though general cancellation of threads is blocked.

The only workaround for this problem is to add additional synchronization to insure that no cancels are prending prior to the call of pthread_cond_wait(), pthread_join(), and pthread_cond_timedwait() and that cancels do not occur during the wait.

7473 PTHREADS Condition not cleared when waiting pthread canceled asynchronously

When trying to destroy a condition variable that was previously being waited on by a thread that was canceled while waiting on the condition, the pthread_cond_destroy() function fails to clear the condition and issues the message "Device busy." This function only fails when the waiting thread is canceled asynchronously. There is no workaround for this problem. This same function works properly (canceling the condition variable properly) when the waiting thread is canceled synchronously.

7476 PTHREADS pthread_cond_init(cond) doesn't fail when cond is in use already

The pthread_cond_init() function should return -1 and give EINVAL if the condition variable has one or more threads waiting on it. Currently, this function succeeds (reinitializes the condition variable) and any threads waiting on that condition variable are lost. The waiting threads are left blocked in pthread_cond_*wait() and a cond_signal() to the reinitialized condition variable doesn't wake it. This problem can be avoided by not reinitializing the condition variable.

7529 PTHREADS pthread_cond_timedwait()'s timeout is a little too quick sometimes

The pthread_cond_timedwait() function sometimes (about 50% of the time) doesn't wait long enough before timing out. There is no workaround for this problem; however, the resulting inaccuracy seems to be very small (1-2 milliseconds).

7605 PTHREADS Posting an hrecv() or hsend() in a pthread application may cause it to hang

If an application uses pthreads, posting an hrecv() or hsend() can hang the system if the handler that is called uses a pthread or reentrant call. A workaround for this problem is to not use pthread or reentrant C calls inside the handler routine.

7864 PTHREADS C lib calls across fork() and nx_nfork() can cause race condition

Using C library calls across fork() and nx_nfork() calls in a pthread application can create a race condition. The current implementation of fork() and nx_nfork() only copy the thread calling fork() or nx_nfork() to the new process address space. The library mutex locks are also copied to the new process space. This situation can cause a race condition such that, if a lock was held by one thread before the fork() call and was requested by another thread after the fork() call, the thread in the new process space will be blocked on the lock. The lock will never be released since the thread holding the lock is not in the new process address space. The hang resulting from this situation can be terminated with a CTRL-C.

7941 PTHREADS Concurrent exec()'s may cause EMULATOR EXCEPTION

If more than one thread executes execlp() at the same time, the service node application can hang with EMULATOR EXCEPTION. Use any method of synchronizing the exec()'s so they don't occur so concurrently to solve this problem.

8113 PTHREADS Concurrent fork()'s can hang a pthread application

If several threads in a pthread program make concurrent fork() calls, the program can hang. A CTRL-C will terminate the hung program. To avoid this problem, do not spawn threads that make concurrent fork() calls.

8136 PTHREADS SIGCONT does not always restart all pthreads

The SIGCONT signal does not always restart the pthreads of a process that have been suspended with the SIGSTOP signal. There is no workaround for this problem.

8146 PTHREADS Threads sometimes hang on a sleep() call

In a pthread application, threads that are suspended temporarily with a sleep() call sometimes become hung and do not reactivate after the sleep time has expired. There is no workaround for this problem. A CTRL-C will terminate the program, if one or more thread hangs in this manner.

8150 PTHREADS pthread_cond_wait() sometimes hangs after completion of pthread_cond_signal()

When a pthread_cond_wait() call is made from the main thread of a pthread application, the call sometimes hangs, even after another pthread has issued a pthread_cond_signal() call to unblock the main thread. This bug is apparently the result of a race condition. To solve this problem, the thread calling pthread_cond_signal() must use the same mutex that the thread that called pthread_cond_wait() used.

8180 PTHREADS Pthread internal error occurs during a file I/O operation

You cannot cancel a pthread during a file I/O operation such as cread(). Doing so will result in a "pthread internal error in thread_suspend" message and the thread will be left in an indeterminant state.

8430 PTHREADS fscanf() sometimes fails when many threads call it concurrently

When many threads within a pthread application call fscanf() concurrently (each thread reading a different file), fscanf() sometimes fails. There is no workaround for this problem other than to avoid having too many threads call fscanf() concurrently.

8730 PTHREADS sigwait() only works with asynchronously delivered signals

The sigwait() function does not work when signals are delivered synchronously; it only works with asynchronous signals. There is no workaround for this problem.

9400 PTHREADS — myptype() returns error when reentrant C library is used

When using the reentrant C library, a myptype() call return "Error 0" after nx_initve() and setptype() calls are made. There is no workaround for this problem.

9401 PTHREADS — Reentrant version of fprintf() causes jumbled output on stderr if numnodes > 1.

8132 TCP/IP — Continuous, heavy network activity can over time panic the system.

System panics have been documented in system configurations where network ports handle heavy traffic (such as through TCP/IP or ftp data transmissions) for periods of several hours. This problem occurs when running both single and multiple network servers. There is no workaround for this problem; if this occurs, you must reboot the system.

7952 VSOCKET — netstat -r does not print full NETDEV names of all interfaces in routing table

The command netstat -r currently only prints the full NETDEV name (e.g. <3>em0) in the routing table when the network interface is not local to the netserver's routing table.

8059 VSOCKET — netstat -r randomly selects the routing table to display

When multiple netservers are configured, netstat -r randomly selects and displays one of the routing tables from one of the netservers. Depending on which netserver's routing table is selected, different routing table entries are displayed. There is no workaround for this problem.

Fixed Bug List

The following lists the bugs fixed since Release 1.1 of the Paragon system software:

6936 BOOT PROCESS	Should bit bucket processor port of MRC on slots marked EMPTY in SYSCONFIG.TXT
7055 BOOT PROCESS	After R1.1 diagnostics are installed, "reset autocfg" doesn't work
7296 BOOT PROCESS	bootmesh -n no longer works.
7308 BOOT PROCESS	bootpp/bootmesh no longer support DEBUG_NODE
7414 BOOT PROCESS	Reset script doesn't work in WW49
7415 BOOT PROCESS	DEVCONF.TXT Must be present or boot fails.
7499 BOOT PROCESS	Paging trees have stopped working with the R1.1.1 patch!
7556 BOOT PROCESS	fscan output become garbled past cabinet 4
7726 BOOT PROCESS	Booting hangs with "[swapon | bootmesh] already run messages"
7866 BOOT PROCESS	scanio reference in reset script incorrect...will not boot paragon
8213 BOOT PROCESS	SDSC Seeing 30-50% of Reboots Fail Under R1.1.3 (Clarify 12249)
8461 BOOT PROCESS	Need to detect automaitcally boards with the MCP ECO.
8462 BOOT PROCESS	missing device files for paging partitions give no error on boot up
8570 BOOT PROCESS	Watchdog does not reboot if a node panics
8921 GP NODE	BETA: User program causes kernel on node to die
8267 HIPPI	HiPPI SRC channel locks up after cable is removed and then replaced.
9168 HIPPI	unexpected packets for HIPPI cause uninformative printf

7085	LIBNX	iprobe call returns 1 even when receive has been posted
7519	LIBNX	Instrumentation hangs thread in routines that have performd nx_spin_lock().
7736	LIBNX	Should not be able to change an established ptype with setptype() in R1.2
7878	LIBNX	Array getting corrupted by setptype()
8768	LIBNX	Fix for xmsg corruption can cause xmsg fragmentation
6916	LOCUS TNC	Simple test causes panic: ux_server_loop rpc: id=199600, ret=0x1000000a
7634	LOCUS TNC	Double ^C can hang system.
7655	LOCUS TNC	killcube(-1,-1) crashes the system with "panic: pvpop_sigpgrp: bad rpc"
7698	LOCUS TNC	Server deadlock on time_lock hangs system (running PFS SAT)
9031	LOCUS TNC	rfork leaks 2 deadnames per remote process creation
7621	MACS	MACS needs to verify critical files exist and exit if they don't.
8020	MACS	Error communication from daemon to root should go straight to root's prompt.
8596	MACS	Connection to MACS rejected for multiple ethernet system
8750	MACS	ENFORCE option acctkill is ignored by MACS and does not kill jobs.
8760	MACS	ENFORCE flag userkill should not kill if acct's No_kill flag is set.
8773	MACS	Jobs running when allocation is exhausted are not killed
8774	MACS	The ENFORCE argument "userkill acctkill" does not

	perform any control functions
8863 MACS	macadmin allows users to execute and change info only root should change
8864 MACS	Macalloc has undocumented switches identical to macadmin
8902 MACS	macadmin -mca allows "-%" to modify allocation
8923 MACS	macalloc functionality can be accessed by general users
8931 MACS	BETA: MACS accounts job twice if NQS queue is undefined
8964 MACS	MACD looses connection to SMD under certain connditions.
9045 MACS	BETA: startup script does not recognize a reboot, no PARABOOT entry made in log
9071 MACS	Alarm message processing is not working in MACS
9167 MACS	macadmin -t yields big rounding error when specify a big amount
6989 MESH UTILS	Allocator can't handle missing nodes in .compute
7257 MESH UTILS	Executing mkpart commands causes huge leaks (new in R1.2)
7376 MESH UTILS	Multiuser test panics kernel: netipc_vm_map_copy_invoke: kr=11
7430 MESH UTILS	pspart reports "pspart: No such file or directory"
7514 MESH UTILS	pspart shows incorrect PRI values
7558 MESH UTILS	Allocator crashes during gang scheduling test
7861 MESH UTILS	mkpart fails mkpart_nd_random EAT
7948 MESH UTILS	pspart prints random strange times for applications TIME ACTIVE
7991 MESH UTILS	pspart reported time incorrectly

8687	MESH UTILS	Rolled-out job does NOT get roll in after overlapping active job finishes.
6947	MESSAGE PASSING	Assert msgp_nxdat.c line 1497 when l2malloc memory exhausted.
6956	MESSAGE PASSING	message passing between processes on same node is very slow
7062	MESSAGE PASSING	Program panics nodes with assertion failure in msgp_select.c
7241	MESSAGE PASSING	sleep in the main program keeps hrecv handler from firing
7245	MESSAGE PASSING	NAS APPSP bnchmrk hangs when message co-processor on, run ok without
7428	MESSAGE PASSING	NAS || bmark FT hangs on 64 nodes
7467	MESSAGE PASSING	NAS || bmark runs very slow on 64 nodes
7619	MESSAGE PASSING	hrecv() not always receiving messages
7637	MESSAGE PASSING	hrecvx() receives msgs from node outside of nodesel parameter sometimes
7671	MESSAGE PASSING	Bandwidth does not meet 75 MB/sec performance goal for R1.2
7770	MESSAGE PASSING	Uninitialized variable causes memory leaks
7785	MESSAGE PASSING	^C out of application created a root-owned null process in .compute
8001	MESSAGE PASSING	Assert ../libnx/rkmem.c line 355 running MCAT PCCM2 & QCD
8134	MESSAGE PASSING	Uninitialized variable in mcmsg_find_detachee can cause assert.
8412	MESSAGE PASSING	MCAT msg co-processor trap caused by bad mt arg in provide call
8850	MESSAGE PASSING	messages sent with isend are received out of order
8885	MESSAGE PASSING	Message Passing Bandwidth below 75MB/sec requirement.

9181 MESSAGE PASSING	BETA: hrecv code crashes system with assertion in file ".../msgp_hwr2.c"
9225 MESSAGE PASSING	Global sends may fail if hrecv handler interrupts the global send
8067 NFS	Open/iwrite/delete on 60 shared files in fortran over NFS hangs the system ...
6869 NQS	Cannot use Paragon interactively in daytime, batch in nighttime.
8087 NQS	QSTAT only showing first 44 jobs
8088 NQS	NQS Time limits not always enforced.
8325 NQS	qmgr move request isn't finding job to move to new queue
8346 NQS	pipe q entries only partial; causes qstat misdisplay and qmgr hiccups
8597 NQS	Connection to NQS rejected for multiple ethernet system
8599 NQS	The NQS needs to be modified to do 2 layer timesharing
8600 NQS	The "np_overrun" implementation does not handle all non-prime case.
8601 NQS	The scheduling job priority is sorted using fixed point arith.
8802 NQS	BETA: NQS can't seem to read .partinfo files for open partitions
8952 NQS	The /sbin/init.d/nqs script should use qmgr to kill nqs properly
9010 NQS	NQS qsub -c swtich only validates with MACS, doesn't set account
6882 OSF COMMANDS	shell script using sed doesn't finish, can't kill
7105 OSF COMMANDS	"/sbin/init s" does not disable user logins

7107	OSF COMMANDS	"quotacheck -a" causes core dump
7171	OSF COMMANDS	dump program rewinds the tape even if "-N" is used
8756	OSF COMMANDS	ls -luR always resets the file modification time on pfs files
9076	OSF COMMANDS	BETA: inetd core dumps as a result of talk and ntalk using same port (517/udp)
8499	OSF DOC	The SPV Deamon hangs during boot when "ALTOS" is used & paging trees turned on.
7275	OSF LIBS	dcos gives wrong answer when pipelined and with large negative input value
7625	OSF LIBS	r13 is modified in ieee and noieee asinf.s causes pccm2 to core dump
7968	OSF LIBS	very slow memset() in libc causes calloc() to be too slow
7990	OSF LIBS	printf does not properly format floating point numbers
8465	OSF LIBS	printf prints wrong values
6875	OSF MICROKERNEL	panic: mf_get_window.inode_pager_setup during tar
6878	OSF MICROKERNEL	Can't get more than 11MB space on 32MB nodes.. System dies
6926	OSF MICROKERNEL	The vm_copy() function is not unwiring pages.
6942	OSF MICROKERNEL	NAS || APPSP hangs on 204 nodes... all-to-all communication (system dies)
6946	OSF MICROKERNEL	Assert in msgp_ipc.c line 203 because MAX_RTS_REX to low for L75
7082	OSF MICROKERNEL	Simple code using message passing and pfs hangs when run with plk
7098	OSF MICROKERNEL	PFS SAT fails to make progress.
7142	OSF MICROKERNEL	MCAT pccm2 panics when writing 512 large history files to UFS.

7262	OSF MICROKERNEL	Data breakpoint fails to generate a SIGTRAP.
7502	OSF MICROKERNEL	MCAT: pccm2 causes panic: norma_deliver_kmsg: netipc_assembly_wrapper
7740	OSF MICROKERNEL	fscan can hang system
7749	OSF MICROKERNEL	MDC with code 0xE9 causes bogus error messages.
7799	OSF MICROKERNEL	port leak leads to kernel assert running 6x32 MUNOPS
8226	OSF MICROKERNEL	Taking a break point trap in dual instruction mode does not work correctly.
8294	OSF MICROKERNEL	panic (norma_ipc_stransit_wait) on compute node when running space-sharing tests
8801	OSF MICROKERNEL	Panic in Fast Out-Of-Line reading a PFS file
9187	OSF MICROKERNEL	TENSOR CAT: irecv losing messages
6873	OSF SERVERS	Occasionally files are truncated to 0 length when system crashes.
7189	OSF SERVERS	nohup... & ; exit panics the system
7475	OSF SERVERS	Improper loads under ipd R1.1.1 hang paragon
7660	OSF SERVERS	All of the intr_delivery() calls in emulator/emul_callback.c are missing the T
7683	OSF SERVERS	R1.2 needs support for propagated TZ among nodes
7684	OSF SERVERS	Dereferencing an uninitialized pointer crashes the system.
7705	OSF SERVERS	lint discovered variable used before initialized in nx_tam_wait()
7767	OSF SERVERS	null process that can't be killed.. repeatable example.
7782	OSF SERVERS	Lite server does not setup an exception handler.
7932	OSF SERVERS	[node: 3] panic: catch_exception_raise when nx/service/.partinfo empty

8040	OSF SERVERS	panic: ux_server_loop rpc: id=11183, ret=0x1000000a (running kenbus1
8054	OSF SERVERS	catch_exception_raise triggered by: "write failed, file system is full"
8084	OSF SERVERS	Controlling process may reap exiting process out from under the TAM
8101	OSF SERVERS	concurrent sleep()'s cause hang on exit (emulator panics)
8106	OSF SERVERS	Number of shared memory segments too low (NRAD can't run apps)
8178	OSF SERVERS	catch_exception_raise panic in dpvpop_nx_get_p_flag() at shutdown
8179	OSF SERVERS	Vsx test fails due to Unix pipe read not returning EINTR on signal.
8242	OSF SERVERS	panic: ux_server_loop rpc: id=11102, ret=0x1000480b
8351	OSF SERVERS	The server gets an exception after unmounting and mounting filesystems.
8421	OSF SERVERS	server dies with VPROC: NULL panic in table()
8426	OSF SERVERS	panic: vio_device_read_synchronous vio_device_read
8520	OSF SERVERS	(SAT/MUNOPS) panic: Master still held
8724	OSF SERVERS	A user code causes the server to panic in get_vnode_port ().
8734	OSF SERVERS	BETA: ^C pthread app caused server exception at psig1()+f8
8780	OSF SERVERS	BETA: panic: ux_server_loop rpc: id=199320, ret=0x1000000c
9069	OSF SERVERS	PFS SAT hangs in credentials_deregister() call
9196	OSF SERVERS	ipd is unable to load NX programs
8168	PAGING TREES	Automatic paging tree setup fails for ENET only